REPUBLIC OF TÜRKİYE

ALTINBAŞ UNIVERSITY

Institute of Graduate Studies

Information Technologies

MALWARE CLASSIFICATION USING DEEP LEARNING

Mohammed Qusy Abd Alkader ALCHALABI

Master's Thesis

Supervisor

Asst. Prof. Dr. Sefer KURNAZ

Istanbul, 2023

MALWARE CLASSIFICATION USING DEEP LEARNING

Mohammed Qusy Abd Alkader ALCHALABI

Information Technologies

Master’s Thesis

ALTINBAŞ UNIVERSITY

2023

The thesis titled MALWARE CLASSIFICATION USING DEEP LEARNING prepared by MOHAMMED QUSY ALCHALABI and submitted on 11 / 4 / 2023 has been **accepted unanimously** for the degree of Master science in Information Technology.

Asst. Prof. Dr. SEFER KURNAZ

Supervisor

Thesis Defense Committee Members:

Asst. Prof. Dr. SEFER KURNAZ	Department of Computer Engineering, Altinbas University	_____
Asst. Prof. Dr. Abdullah Abdu IBRAHIM	Department of computer Engineering Altinbas University	_____
Asst. Prof. Dr. Zeynep Altan	Department of Software Engineering, Beykent University	_____

I hereby declare that this thesis meets all format and submission requirements of a Master's thesis

Submission date of the thesis to Institute of Graduate Studies: ___/___/___

I hereby declare that all information data presented in this graduation project has been obtained in full accordance with academic rules and ethical conduct. I also declare all unoriginal materials and conclusions have been cited in the text and all references mentioned in the Reference List have been cited in the text, and vice versa as required by the abovementioned rules and conduct.

Mohammed Qusy Abd Alkader ALCHALABI

Signature

ACKNOWLEDGEMENTS

My first thanks and gratitude to God who gave me the strength, ability, and guidance to complete my master's thesis. I also pray God to have mercy on my father’s soul, who wanted to see me complete my master's study. I would also like to extend my thanks and appreciation to my mother for her continuous encouragement and support for me to reach this stage. Special thanks to my wife for her unwavering love and support. She was always there for me, offering words of encouragement and helping me stay on track. Many Thanks are extended to my children who stood by me during this journey. And to my big brother, from whom I learned a lot to accomplish my thesis. And to my sister who stood with me throughout the study period. Finally, my thanks and gratitude to all my friends and relatives who supported me. Thank you all from the bottom of my heart.

Also, I would also like to extend my sincere thanks and appreciation to my advisor Prof. Asst. Dr. SEFER KURNAZ for his precise guidance and follow-up to complete the master's thesis.

I would also like to acknowledge my university, I am grateful for the opportunity to attend such an esteemed institution, and I am grateful for the support and guidance I have received from the university throughout my academic journey.

ABSTRACT

MALWARE CLASSIFICATION USING DEEP LEARNING

ALCHALABI, Mohammed Qusey

M.Sc., Information Technology Altınbaş University

Supervisor: Supervisor: Asst. Prof. Dr. Sefer KURNAZ

Date: April / 2023

Pages: 60

In this study, the performance of different machine learning algorithms was evaluated for detecting malware based on the permissions used by Android applications. The study aimed to compare the performance of four different algorithms: k-nearest neighbors (KNN), a convolutional neural network (CNN), a convolutional neural network with long short-term memory (CNN-LSTM), and a convolutional neural network with the gated recurrent unit (CNN-GRU) by using the Python programming language. The dataset used in the study consisted of binary vectors representing the permissions used by each application and was labeled as either malware or non-malware (benign).

The performance of the algorithms was evaluated using standard metrics such as ACC, PREC, and recall. The results showed that the (CNN-GRU) algorithm had the highest accuracy with 95%, followed by the (CNN) algorithm with 91%. The (KNN) algorithm had an accuracy of 87% while the (CNN-LSTM) algorithm had an accuracy of 90%. All four algorithms had similar precision and recall scores, with the (CNN-GRU) algorithm having the highest F1-score of 94%. The loss values for all four algorithms were also calculated and compared.

In conclusion, the results indicate that the (CNN-GRU) algorithm performed best in terms of accuracy and F1-score, followed by the (CNN) algorithm. These findings can help inform future research in the field of malware classification using deep learning.

Keywords: Malware Classification, Deep Learning, Machine Learning, CNN, RNN.

TABLE OF CONTENTS

pages

LIST OF TABLE

pages

LIST OF FIGURES

Pages

ABBREVIATIONS

ML	:	Machine Leaning
DL	:	Deep Learning
FUN	:	Function
ACC	:	Accuracy
PREC	:	Precision
REC	:	Recall
F1-S	:	F1-score

1. INTRODUCTION

1.1 INTRODUCTION

In this chapter, we will introduce the concept of malware and the importance of its classification in ensuring the security of computer systems and networks. We will discuss the limitations of traditional methods for detecting and classifying malware and the potential of DL techniques in overcoming these limitations.

1.2 RESEARCH REVIEW

Malicious software that has been inappropriately installed or modified by a hacker or computer user is known as malware. Malware may seriously harm computers and the data they contain, even if it might not be detected as such. The most prevalent types of malicious software include viruses, worms, Trojan horses, and spyware (i.e., software that monitors user activity).

Malicious samples have been studied using malware analysis to comprehend their behavior and how they change over time. Malware analysis aims to identify new characteristics of malware that may be utilized to strengthen security controls and make evasion more challenging.

The ability of authors of harmful code to evade detection by utilizing strategies like obfuscation, polymorphism, and metamorphism is one of the toughest hurdles in malware analysis. These methods alter a malware's binary and hash, making it more evasive to detection. They do not alter the dangerous program's behavior, though, thus criteria based on MD5 hashes can still classify them as threats. Researchers are increasingly using ML to automatically extract knowledge from malware samples in order to get around this problem [1]. Since ML can automatically spot patterns and characteristics in data that are hard for people to spot, it makes sense to use it to support the knowledge extraction process. This methodology has been used in several literary works, and while methods and outcomes vary greatly, all rely on computer-generated models.

Malware analysis often aims to identify new, hard-to-evade characteristics of malicious software, including semantics, and then utilize these characteristics to inform the development of more effective security solutions. ML is now employed in the field of

malware analysis and is capable of extracting useful knowledge from massive datasets of malware data [2].

Traditional malware classification [3, 4, 5, 6, 7] techniques frequently rely on labor-intensive, error-prone human feature development. However, the necessity for reverse engineering which may be challenging and time-consuming—limits the use of dynamic analysis, while sometimes helpful, can only look at one particular execution route, making it challenging to completely comprehend virus behavior.

The fact that existing malware classification techniques are unable to keep up with the diversity and quantity of emerging dangerous software versions is one of its shortcomings. It has been shown to be beneficial to apply ML techniques, however, how effectively these computer-based classification algorithms are able to differentiate between various varieties of malware has a significant impact on their efficacy. Due to the frequent appearance of new versions, virus writers may easily avoid detection by antivirus software by making minor modifications to their code and packaging them with a new name or icon.

Although many of the current approaches for classifying malware are efficient, they have drawbacks that make it harder and harder to classify new harmful software versions [8]. DL has become an effective way to classify malware because it is better than traditional methods [9, 10, 11, 12] at identifying the features of malware and analyzing large amounts of data. However, imaging-based and DL methods of classifying malware rely on a process that changes the binary code embedded in each sample into grayscale images prior to analysis. This conversion can negatively impact the classification of ACC.

When gray images are converted to a uniform size, the original binary information is partially missing and this can result in a lower ACC rate. Secondly, during the conversion process, a lot of redundant information can be created that isn't conducive to malware classification. This further lowers the ACC rate.

Furthermore, because the conversion process is imperfect and some binary bytes are not conducive to malware classification, redundant information can be created. This further lowers the ACC rate.

To improve the performance of the categorization of malware using photos, it is necessary to rely on large datasets that include labeled training samples. Other approaches, such as improving the conversion process and reducing redundant information, can also lead to better classification results [8]. Figure 1.1 shows the converting malware file to image.

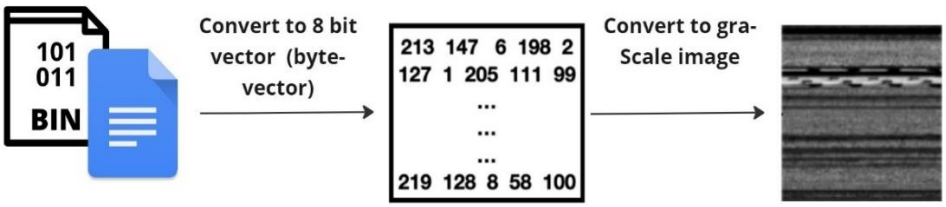

Figure 1.1: Malware File to Image.

1.3 METHODOLOGY

The proliferation of malware threatens computer systems and networks. Malware, short for malicious software, is software meant to sabotage or abuse computer systems. It might appear as malwares such as Trojans, worms, viruses, and more. These malicious programs can steal sensitive information, disrupt operations, and cause financial losses. As a result, malware detection and classification are a critical task for the security of computer systems and networks.

Historically, approaches for identifying and classifying malware have relied on signature-based detection or heuristic-based detection. Signature-based detection looks for specific programmed

instructions in the virus and compares them to a database of known virus signatures. Heuristic-based detection, on the other hand, tries to discern patterns from already known viruses and uses them to identify new variants. However, these methods have limitations when faced with new and unknown variants of malware.

Recently, DL approaches have been applied in various areas of computer science and have shown promising results. The use of DL in malware classification is a relatively new field of research, but it is expected to improve the ACC and effectiveness of malware classification. DL-based malware classification systems have the potential to automatically learn features and patterns from malware samples, making them less dependent on human-engineered features. Additionally, DL-based systems can adapt to new variants of malware and improve their detection performance over time. However, little research has been

conducted into the effectiveness of DL in malware classification and how well these techniques perform compared to traditional methods is unclear.

1.4 EXPECTED RESULTS

The proliferation of malware threatens computer systems and networks. Traditional methods for fighting it, such as signature-based detection (where you look for specific programmed instructions in the virus) or heuristic-based detection (where you try to discern patterns from already known viruses), have limitations when faced with new and unknown variants. The use of DL techniques in computer science has been successful in various areas and is expected to improve the ACC and the effectiveness of malware classification.

However, little research has been conducted into the effectiveness of DL in malware classification. How well these techniques perform compared to traditional methods is unclear. This research addresses the problem of creating a DL system capable of accurately classifying different types of malware. This research aims to evaluate DL-based malware classification systems and identify any limitations with the ultimate goal of developing a robust, intelligent security system that can improve computer network protection.

This research may also explore different architectures, algorithms and techniques to use in DL-based malware classification systems, as well as techniques to improve the generalization and robustness of the systems against adversarial samples. Additionally, this research may also focus on the development of an efficient and effective method for collecting and annotating large datasets of malware samples for training DL based malware classification systems.

Overall, this research proposal aims to fill the gap in knowledge on the effectiveness of DL in malware classification and to develop a more effective and robust security system for computer network protection. The proposed research aims to investigate the effectiveness of DL techniques for classifying different types of malwares and evaluate their performance compared to traditional ML techniques.

The research objectives include:

a. Investigate the potential effectiveness of deep learning (DL) techniques, such as convolutional neural networks (CNN), CNN-LSTM, and CNN-GRU, in accurately classifying various types of malwares.

b. Evaluate the performance of different DL architectures, including CNN, CNN-LSTM, and CNN-GRU, and traditional machine learning (ML) techniques, such as k-nearest neighbors (KNN), for malware classification.
c. Compare the effectiveness of DL-based malware classification with traditional ML techniques, including KNN, to determine the advantages and disadvantages of each approach.
d. Analyze the impact of different feature representations on the accuracy of DL-based malware classification to understand the effect of different feature representations on the classification performance.
e. Investigate the generalizability of DL/ML-based malware classification models by testing them on diverse datasets and malware samples to identify any limitations in their generalization capabilities.

1.5 THESIS ORGANIZATION

The remainder of this study is divided into the following chapters:

a. The second chapter: most recent research in this area.

b. The third chapter: a detailed explanation of the technique, the system-building phases, and the procedures that will be included in the dataset, including the pre-processing stage, the models, the cooperative study of the models, and the last stage, the training stage.

c. The fourth chapter: provides the final training results for the model. Apply assessment tools, then compare the results.

d. The fifth chapter, which is the conclusion, provides an overview of all the work that will be done in our thesis, including the selection of the models, the phases, and the outcomes.

1.6 CONCLUSION

In conclusion, this chapter has introduced the concept of malware and highlighted the importance of its classification in maintaining the security of computer systems and networks. Traditional methods for detecting and classifying malware have limitations, but

DL techniques have the potential to overcome these limitations. In the next chapter, we will delve deeper into the topic by introducing the architecture of DL and discussing related work in this field.

2. LITERATURE REVIEW

2.1 INTRODUCTION

As we've already stated, our objective is to create DL/ML models that are able to recognize malware with high ACC and low false positive rates. The literature contains several papers on the use of DL/ML methods for malware classification. The sections will mention giving an overview of the various DL/ML classifiers used and the performance metrics utilized to assess the performance of said classifiers in addition to a summary of the various DL/ML-based malware classification methods that exist in the literature.

2.2 RELATED WORK

Both the early efforts SAVE [13] and SAFE [14], which served as inspiration for many other researchers, employed heuristic static malware detection techniques.

For finding harmful code in executable files, pattern matching has been a common technique. In contrast to subsequent approaches, which were designed to examine both the header and body of portable executables (PE), earlier publications advocated employing patterns to identify malicious code [15]. Malware in software products is commonly concealed via packing and obfuscation [16]. Without assuming anything about the methods employed to make it difficult for humans to understand, recent publications have tackled the issue of decoding obfuscated malware [17].

Strand is a gene sequence classifier that can easily handle unstructured input such as source code or machine code, and it was utilized by researchers in [18]. To show Stand's suitability for categorizing malware, they utilized the dataset. They were able to achieve ACC levels of 95% and higher with less training time than our model required.

Opcodes from executables were utilized by authors in [16] to identify malware using data mining techniques. For the categorization of malware, the study in [19] suggested using word2vec, a freshly created and well-liked program to evaluate natural language documents. Word vectors created by Word2vec are then used to categorize texts by creating word embeddings for each word in the text. Popov passed machine instructions through the Capstone disassembler, which produced the opcodes, and then used word2vec to create word embeddings from the malware opcodes. Then, a classifier based on convolutional neural networks is used to categorize executable programs using these word vectors. A modest

dataset of 2400 (PE) from just two classes—malicious and benign—is used to evaluate the system, with up to a 97% success rate.

Traditional approaches for classifying malware relied mostly on static or dynamic analysis to gather information [4]. After that, categorization was done using ML methods. The malware analysis approach based on gray pictures of malware binaries was initially proposed by [20] in relation to the categorization of malware images. The grayscale pictures of the binary files were transformed, and the texture of gist characteristics were then extracted for malware classification.

Traditional malware image classification techniques have been increasingly supplanted by DL methods have demonstrated considerable benefits for large-scale picture classification applications [21] [22]. As a result, it examines the relevant research on malware classification from two key perspectives: approaches based on feature extraction and ML, and methods based on malware pictures and DL.

Malware classification techniques based on ML may be separated into two categories: static features, and dynamic features, depending on the various feature extraction techniques. Another type of static feature is the texture included in virus pictures.

2.2.1 Static-Features

The majority of these techniques examine the file structure, bytecode, and disassembly code. The Static analysis provides the benefits of quick speed and great efficiency without real execution. There have been several research on the categorization of malware using static characteristics. For instance, [7] provided a tutorial on how various ML techniques can be used in the extraction and analysis of a variety of static characteristics of PE binaries and conducted an extensive survey of various ML methods for classifying static characteristics of PE files.

[23] created the labeled benchmark dataset EMBER to train ML models for statically analyzing malware. The dataset contained characteristics that were taken from 1.1M PE binaries: 9K training samples (3K malicious, 3K benign, and 3K unlabeled), and 2K test samples (1K malicious, 1K benign).

[24] deconstructed files and collected static characteristics from malware binary files. This approach categorized 9 malware families with a 99% ACC rate using multi-dimensional characteristics. Regarding techniques based on malware image texture features, it transforms

the issue the issue of picture classification by malware classification.

For Exp, [8] developed a system that transforms disassembly data into gray pictures in order to detect and categorize malware. The local mean approach was used to compress and map the grayscale pictures to feature vectors. An ensemble learning technique based on k-means and diversity selection was suggested to categorize malware.

Malware binary data were transformed into gray pictures and entropy graphs as part of the malware classification approach [9] suggested. The results of the trial demonstrated that their technology can accurately identify malware families. However, the issue of reverse analysis frequently arises when collecting static characteristics from malware disassembled code. If malware disassembly code couldn't be obtained, the technique based on static characteristics would be invalid.

2.2.2 Dynamic-Features

The automatic execution of malware in a sandbox or other physical environment control during dynamic analysis in order to study its behavior. When the static analysis is ineffective, malware activities is directly examined via dynamic one. Because it captures the real malware runtime actions, dynamic one provides the advantages of a low false alarm rate and a high ACC rate.

For instance, [9] described a technique for classifying malware based on methods for grouping flow characteristics and sequence alignment. They discovered the best technique for categorizing malware based on network activity similarities using real malware traffic.

[5] provided a way to construct the feature vector for malware utilizing behaviors of run-time and online ML methods classification. The ACC rates for training and testing were 94% and 93%, respectively, according to their experimental findings.

[25] put up an NLP approach API sequences modeling in 2018. The characteristics were retrieved using the n-gram, word bag, as well as TF-IDF models. Linear SVM was then used to classify the malware, with a 95% ACC rate.

Using the findings of the Cuckoo Sandbox, [6] developed a malware family categorization method for 11 harmful families by extracting their salient API characteristics. The system uses the classifiers RF, KNN, and DT to categorize harmful software.

[3] recorded API calls performed by the Windows operating system and then examined 7107 distinct dangerous programs from diverse families, including viruses, backdoors, and trojans, in a sandbox environment that was completely isolated from the outside world.
The findings of the sequential analysis were then converted into a format that allows for the usage of various categorization algorithms and techniques. Although dynamic analysis is more accurate, it can only study one particular execution path's behavior; it is challenging to traverse the behavior traits of all the execution pathways. Dynamic analysis takes time, though. As a result, it is challenging to handle the malware varieties' fast expansion.

2.2.3 Other Works

The authors describe a unique method for detecting Android malware in this literature review [74] that makes use of a multi-class malware detection system powered by Cuda that is based on Gated Recurrent Units (GRU). The method is assessed using up-to-date datasets from Android applications (AMD and Androzoo) and contrasted with benchmark techniques and artificially generated DL architectures. The findings demonstrate that the GRU-based system excels with a detection ACC of 98.99% while yet retaining a respectable level of speed efficiency.
In the study [75], MAPAS, a malware detection system that analyzes dangerous app behavior based on API call graphs using CNN, is introduced. Instead of building a classifier model, MAPAS employs CNN for feature identification and a lightweight classifier for similarity computation. The prototype performed better than a leading system in tests, with higher ACC, a faster classification speed, and less memory use.
The research [76], report employs ML and reverse-engineered Android app features to find app security flaws. The research offers a model that uses ensemble learning methods for increased ACC and a huge dataset of malware samples. The results indicate a 96.2% ACC rate for malware detection and a 0.3 False Positive Rate. The model includes overlooked elements and uses a single reverse-engineered feature as an input during training.
This paper [78], does a critical analysis of earlier studies that employed ML to identify Android malware. The evaluation includes supervised, unsupervised, DL, and online learning approaches and classifies them based on whether static, dynamic, or hybrid characteristics are used.

This paper [79] does a critical analysis of earlier studies that employed ML to identify Android malware. The evaluation includes supervised, unsupervised, DL, and online learning approaches and classifies them based on whether static, dynamic, or hybrid characteristics are used.

This study [80] suggests a behavior-based methodology for predicting Android virus odors. The model includes a mix of static and dynamic features that are organized into a small number of cluster classes to streamline the intricate feature relationships. LSTM and ensemble ML models are used to classify the cluster sequence snapshots. The model performed consistently well when predicting new sequences with high ACC when it was tested against popular ransomware assaults. The suggested remedy offers a warning before damage is done in order to stop malicious payloads.

2.3 ML

2.3.1 Definition

ML [26] is used to train computers to analyze data more effectively. ML is used to find relevant information when data analysis is unable to do so. The demand for machine learning (ML) is expanding along with the availability of datasets since organizations utilize it to extract pertinent information and learn from data. Robot learning without explicit programming has been the subject of experiments, and mathematicians and programmers have used a variety of approaches to tackle this problem with big datasets.

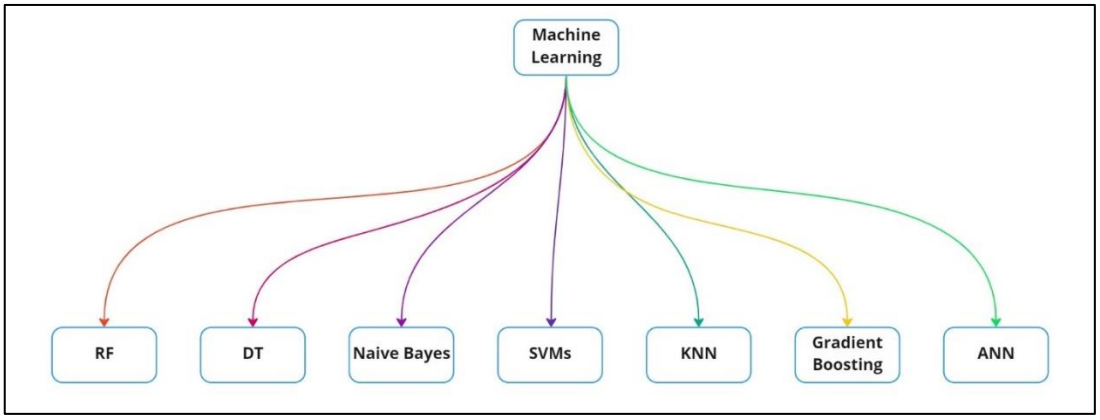

Figure 2.1: ML Method Used in Malware Classification.

2.3.2 Random Forest

The Random Forest Algorithm is a technique of ML that uses separate classifiers that have been trained and aggregates their predictions. It is only utilized for classification or regression issues and is especially designated for decision tree classifiers.

With this technique, a selection of decision trees is recreated from a training sampling subset that is selected at random. The final step is to determine the test item's final class by grouping the votes from several decision trees [27].

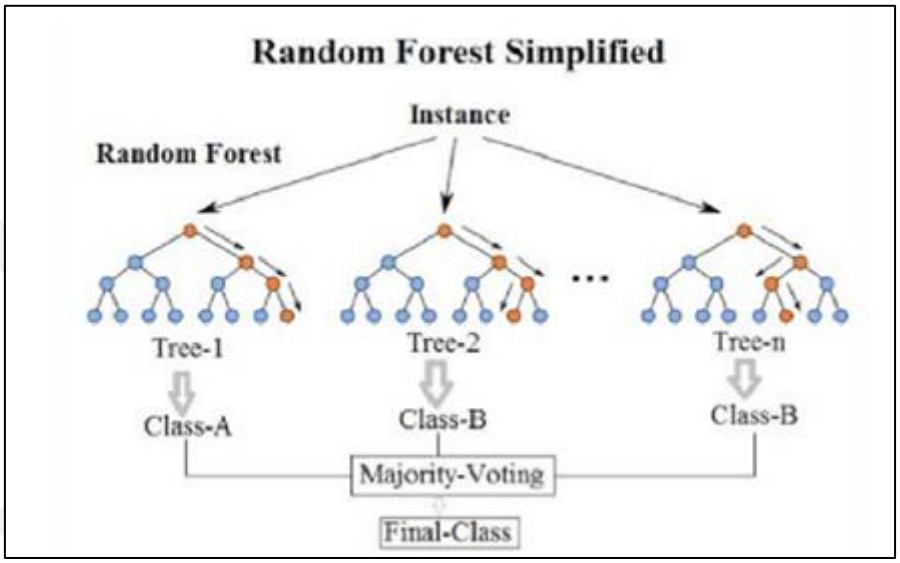

Figure 2.2: Random Forest Algorithm Mechanism. [28]

2.3.3 Decision Tree

Decision trees are a common data mining method for making predictions (See Fig 2.3).

A decision tree consists of nodes and leaves logical diverging points with each node where a certain data feature would be checked and the data would be separated as necessary. The values anticipated for the point are represented by the leaves. Recursion is the foundation of all decision trees. The normal distribution and other underlying assumptions are not used in the construction of decision trees, as well as correlation or collinearity between explanatory factors also can be disregarded [29].

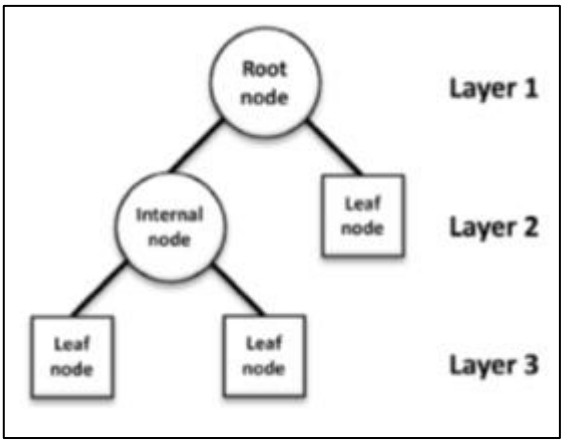

Figure 2.3: Structure of a Decision Tree. [29]

2.3.4 Naive Bayes

One of the first and most well-known ML techniques is the Naive Bayes algorithm [30]. Naive Bayes makes the naive assumption that the variables are independent in order to make the probability-based classification calculation viable. Such an assumption about actual situations is clearly oversimplified. Naive Bayes, however, performs admirably when tested on real databases, especially when paired with an attribute selection technique. As seen in fig 2.4, single classifiers are stacked in a tree inside a hierarchy of Naive Bayes classifiers. After that, a new case proceeds as follows: Beginning at the root node, the instance is categorized by the top-level Naive Bayes classifier; based on the results, a branch is chosen and the instance is categorized once again according to the Naive Bayes model corresponding to that branch, and the final result is returned.

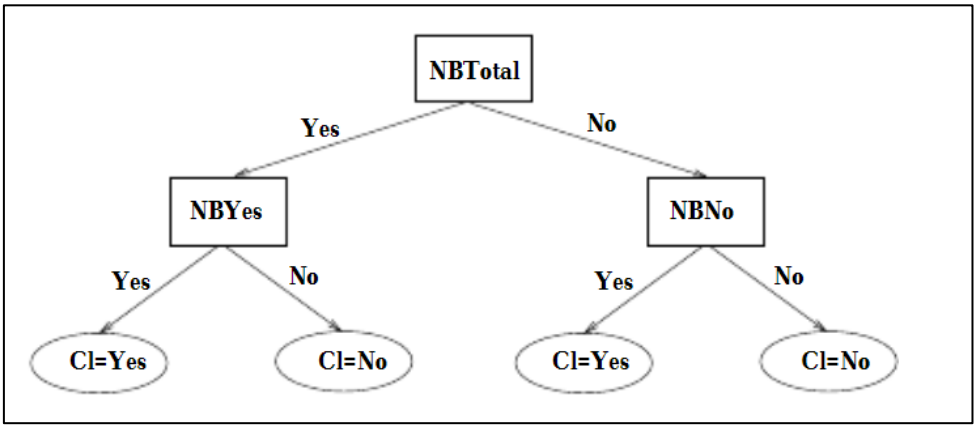

Figure 2.4: Structure of a Naive Bayes. [31]

2.3.5 Support Vector Machine

Many AI techniques based on supervised learning methods may be used to categorize various components. in 1992 the Wide margin separators were created by Boser and his colleagues [32], and Cortes and Vapnik [33] gave them the moniker Support Vector Machines (SVM) in 1995. They are based on the presence of a linear classifier that makes use of kernel FUN s and are grounded in a sound mathematical theory [34]. The primary purpose of support vector machines was to solve binary classification issues. Researchers have proposed approaches for extending SVM to more than two classes because the majority of real-world issues need for a classification with more than two classes.

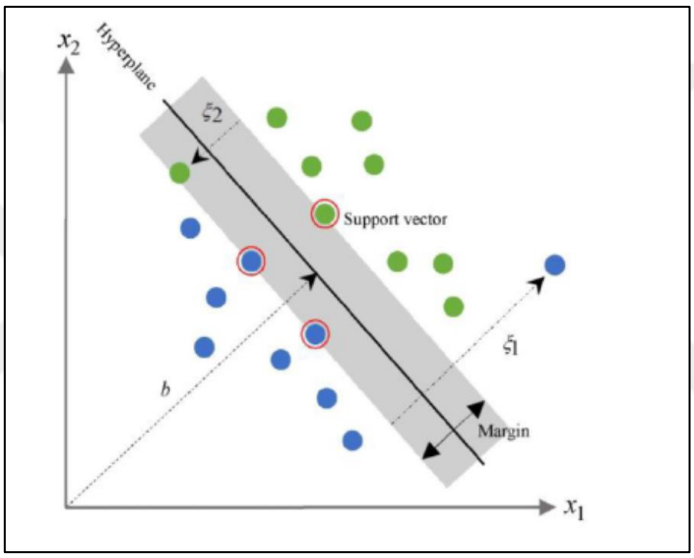

Figure 2.5: Structure of SVM. [35]

2.3.6 K-Nearest Neighbor

The supervised learning method used by KNN necessitates the labeling of data prior to generating predictions. It may be used for grouping and regression. The nearest neighbors are represented by the number K. With no training phase, the KNN algorithm makes predictions based on the Euclidean distance to the KNN. This approach may be used in the breast cancer prediction dataset as benign and malignant tumors have previously been identified. Based on the majority class among its k-nearest neighbors, the label is given. The KNN method is shown in Figure 2.6 [36].

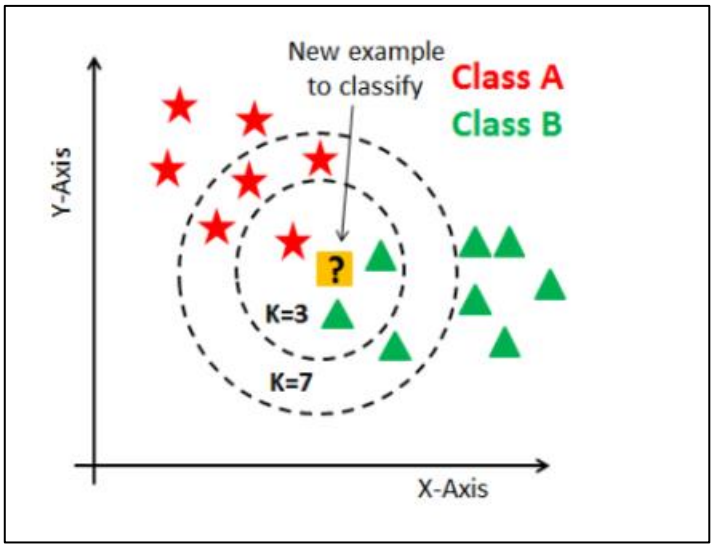

Figure 2.6: Structure of KNN. [36]

2.3.7 Gradient Boosting

Gradient Boosting (GB) is a ML technique that uses a group of weak prediction models to solve classification and regression problems. This method allows for the stage-wise optimization of any differentiable loss FUN, which generalizes models [37]. Leo Breiman had the lightbulb moment of realizing that boosting might be seen as an optimization approach on a suitable cost FUN [38], and Friedman later explicitly developed this idea [39]. This is the origin of the gradient boosting idea. The GB methods may be viewed as an optimization of the cost FUN throughout the space of FUN s by iteratively choosing a FUN (weak hypothesis) that goes in the direction of the negative gradient. This understanding of GB has inspired the creation of boosting techniques outside of regression and classification issues.

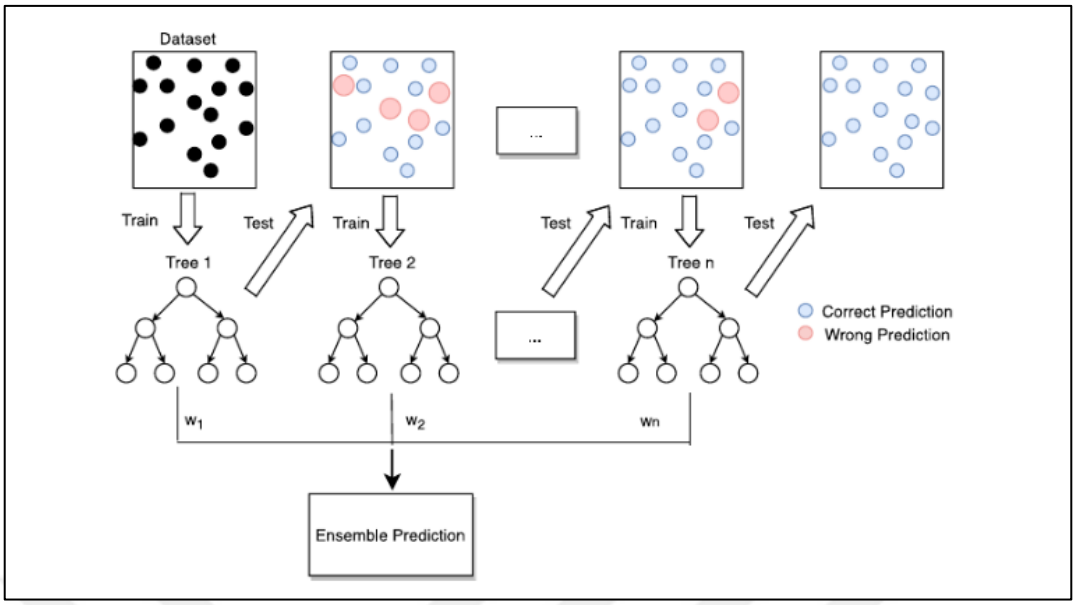

Figure 2.7: Structure of Gradient Boosting. [40]

2.3.8 Artificial Neural Network

A computational model called an ANN is based on the biological nervous system [41]. The structure of an artificial neural network is shown in Fig.2.8. An artificial neural network has three layers: an input layer, a hidden layer, and an output layer. One-dimensional vectors are used by artificial neural networks. Before feeding an image to an artificial neural network, it must only exist in one dimension. An ANN finds the parameters that minimize the error FUN by solving an optimization problem in order to learn.

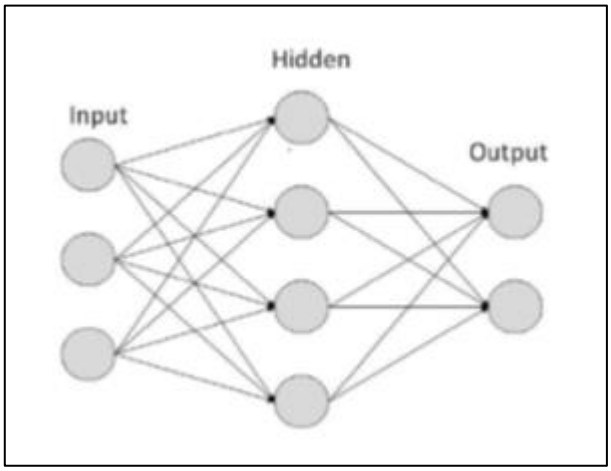

Figure 2.8: Structure of ANN. [42]

2.4 DL

2.4.1 Definition

A branch of ML called DL makes use of hierarchical structures to extract high-level representations from data. Numerous AI fields, including semantic parsing [43], transfer learning [44, 45], NLP [46], computer vision [47, 48], and others, heavily rely on this new methodology. The present expansion of DL is primarily driven by three factors: improved chip processing (such as GPUs), decreasing prices of computing gear, and improvements in ML techniques [49].

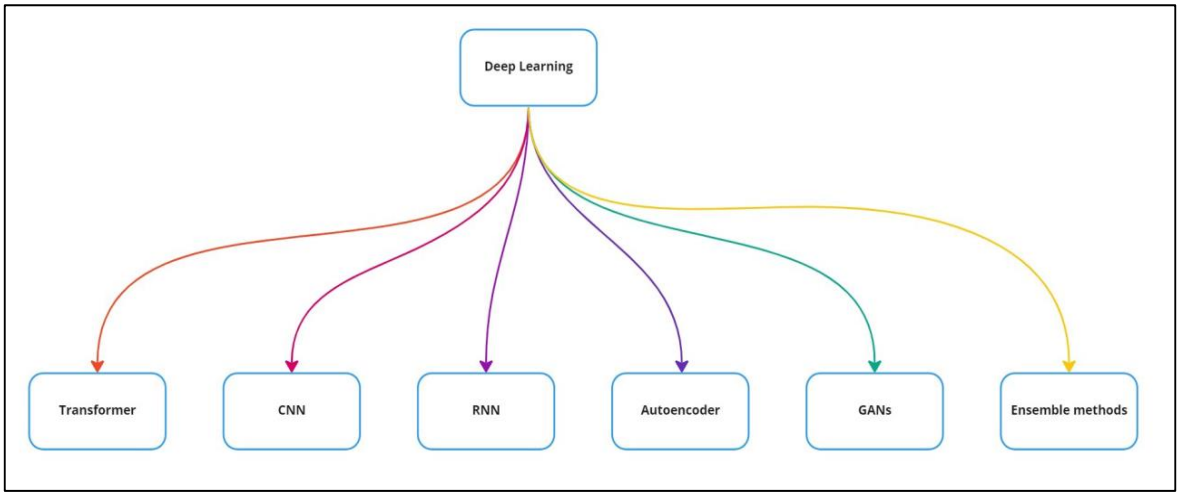

Figure 2.9: DL Method Used in Malware Classification.

2.4.2 Autoencoder

To learn effective encodings, an autoencoder, a more advanced sort of artificial neural network, is utilized [53]. An autoencoder is taught to rebuild its own inputs X, as opposed to conventional neural networks, which are trained to predict a target value Y from inputs X. As a result, the output vectors produced by an autoencoder have the same number of dimensions as the input. An autoencoder's general operation is shown in Figure 2.10. The autoencoder is optimized by lowering the reconstruction error throughout the process, and the learned feature is the appropriate code.

The representative and discriminative characteristics of raw data are typically insufficiently captured by a single layer. To solve this problem, researchers now employ deep autoencoders, which pass the coding learned from one autoencoder to the next. The idea of

a deep autoencoder was first put out by Hinton et al. [54], and it is now the subject of multiple studies [55, 56]. Frequently, a back-propagation approach like the conjugate gradient method is used to train the deep autoencoder. However, if there are mistakes in the first few layers, the model can stop working and learn to use the training average set of data. Pre-training the network with starting weights that roughly reflect the answer is a typical approach to solving this problem [54]. Several autoencoders have also been suggested the representation made as "constant" with respect to changes in input as is practical.

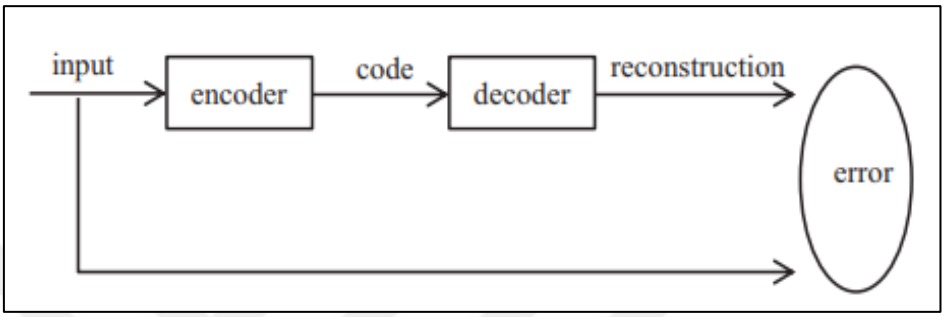

Figure 2.10: The Pipeline of an Autoencoder. [52]

2.4.3 Convolutional Neural Network (CNN)

Given its effectiveness in training several layers, CNN are among the most well-known DL approaches [50]. CNNs are frequently employed in a variety of computer vision applications due to their shown great efficiency. The CNN design's entire pipeline is seen in Fig. 2.11. The three primary neural layers that make up a CNN are fully connected layers, pooling layers, and convolutional layers. Each layer has a distinct FUN. The layer-by-layer composition of a typical CNN architecture [51] for image categorization is shown in Fig. 2.11. Both a forward stage and a backward stage are used to train the network. In the forward step, the input picture is modeled using the weights and biases that are in effect for each layer. The loss cost is calculated using chain rules based on the comparison between the model's prediction and the ground truth labels. During the backward stage, the gradients of each parameter are then determined using these calculated loss costs. These gradients are used to update the parameters, preparing them for the next forward computation cycle. The learning process continues with repeated forward and backward computations until it reaches an appropriate termination point.

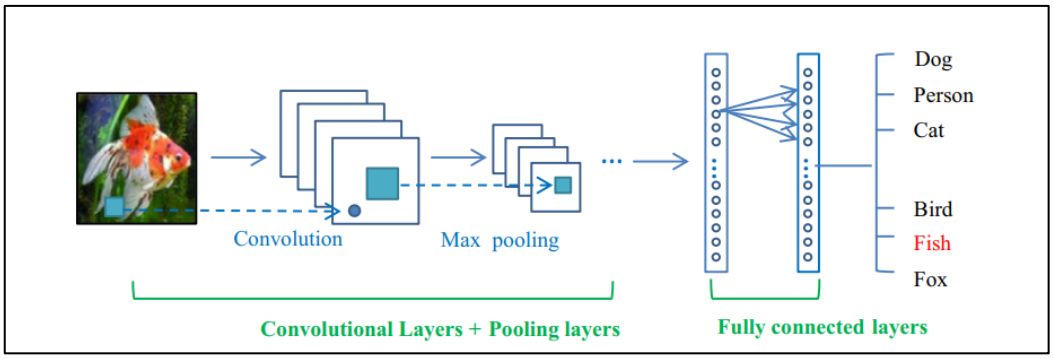

Figure 2.11: The Pipeline of The General CNN Architecture. [52]

2.4.4 Recurrent Neural Network (RNN)

ANNs are nonlinear models that drew inspiration from the biological brain's neural network. The output *yi* of each node *i* in the directed graph of the ANN, which is made up of linked nodes, is produced by applying a transfer FUN *fi* [57] (such as sigmoid or Heaviside). There is a connection weight between the nodes as well as a threshold bias. The feedforward ANN is a simple neural network in which information travels from the input to the output while passing through any potential hidden nodes. In order to semantically connect related sentences, it is important to keep in mind that ANNs can readily project a specific lexicon onto hidden nodes [58]. RNNs can employ internal state to process input sequences, but feedforward networks cannot [59]. RNNs, as opposed to feedforward networks, feature cyclic or feedback connections that enable state updates depending on past states and present inputs. RNNs are improved in sequence modeling jobs as a result. RNNs have the benefit of remembering words provided at a previous iteration, which is useful for establishing context in natural language processing (NLP) operations [58]. Fig. 2.12 clearly distinguishes

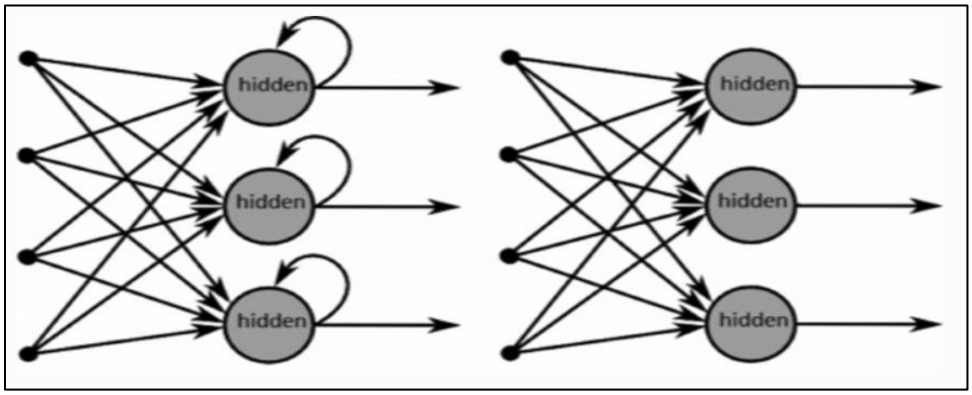

Figure 2.12: RNN (left) vs. Feedforward (Right). [58]

between an RNN network and a FNN, as first depicted by Mulder et al. [58].

Long Short-Term Memory Neural Network

Based on RNN, LSTM has developed and produced favorable outcomes in a variety of applications. Since LSTM has a flexible ability to manage crucial events with extended durations and time series delays, it is particularly well suited for tackling the challenge of time series forecasting. By keeping relevant information and eliminating worthless information, LSTM successfully solves the vanishing gradients issue of the RNN model. Fig 2.13illustrates the LSTM's construction [60].

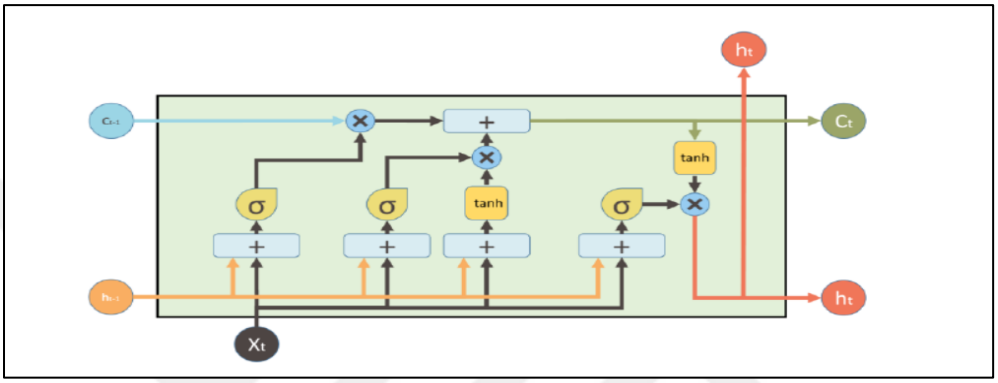

Figure 2.13: The Structure of Long Short-Term Memory Neural Network (LSTM). [60]

Gated Recurrent Unit

The Gated Recurrent Unit (GRU), a potent variation of the standard RNN, uses the combined gating mechanism, like an LSTM, as a short-term memory solution. A system within the GRU called gates controls and even circulates the information flow. The gates assist the GRU cell in learning whether information is crucial to store or remove. As a result, crucial information is passed on to enable prediction Despite having a simpler architecture than a typical RNN, GRU has been demonstrated to be performance and speed efficient. The basic organization of the GRU is shown in Fig. 2.14 [61].

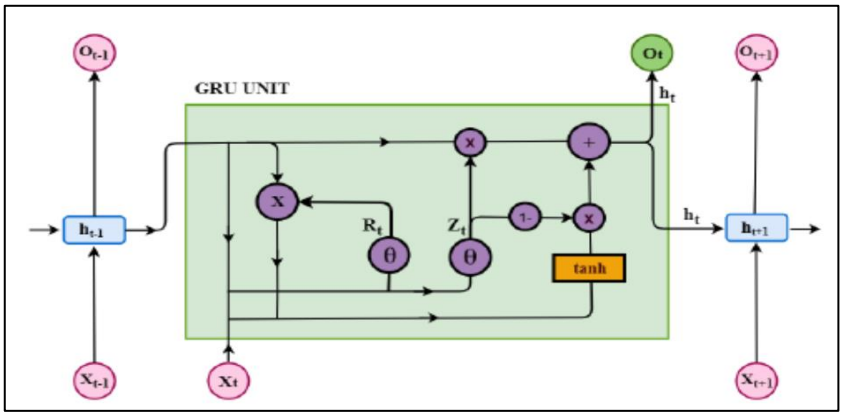

Figure 2.14: The Structure of Gated Recurrent Unit (GRU). [61]

2.4.5 Ensemble Learning

Ensemble learning is a type of ML paradigm that utilizes base learners, commonly referred to as weak learners, that are trained and combined to tackle various real-world problems. EML techniques are also referred to as" metal earning techniques" since" meta-learning" is the term for the process of learning from base learners [62, 63, 64, 65, 66].

Typically, the EML process involves creating and merging foundation learners Fig.2.15. The ensemble generation and base learners are made in the initial stage. The ensemble trimming phase eliminates some of the first step's newly produced FUN s after that.

The last stage combines the selected FUN s using the ensemble combination. EML techniques often fall within a broad family of algorithms that are founded on both solid heuristics and learning-theoretical ideas [67].

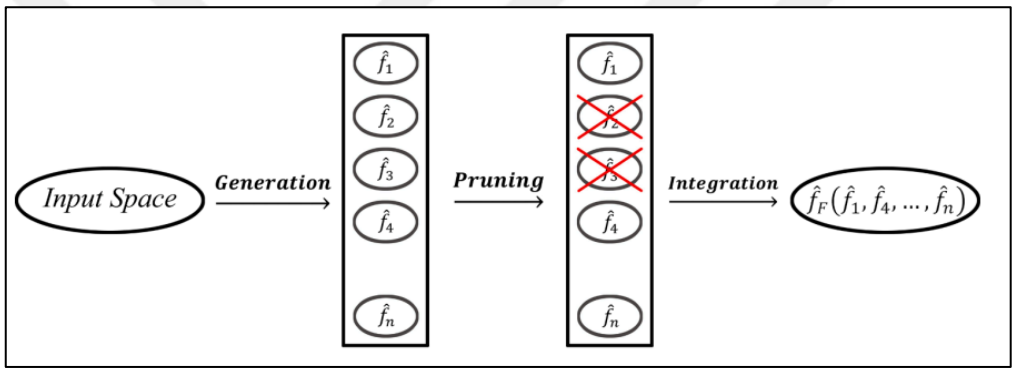

Figure 2.15: Ensemble Learning Architecture. [68]

2.4.6 Generative Adversarial Networks

First, the adversarial learning technique for generative models was presented by Goodfellow et al. [69]. The min-max two-player zero-sum game is the core component of GAN. In this game, one player gains benefits at the expense of the other player's comparable loss.

The participants in this scenario represent the discriminator and generator GANs, which are two distinct networks. Identifying whether a sample is a part of a true distribution or a phony distribution is the discriminator's primary goal. While the generator creates a bogus sample distribution in an effort to trick the discriminator.

Together, a generator (G) and a discriminator (D) create fresh data samples that are comparable to the training data. The generator creates bogus samples to deceive the discriminator, while the discriminator is in charge of detecting if a particular sample is real or phony. With a probability value near to 0.5 for both genuine and fake samples, the

generator's objective is to create samples that are too similar to the actual samples for the discriminator to be able to tell them apart. Fig. 2.16 depicts the overall architecture of a GAN.

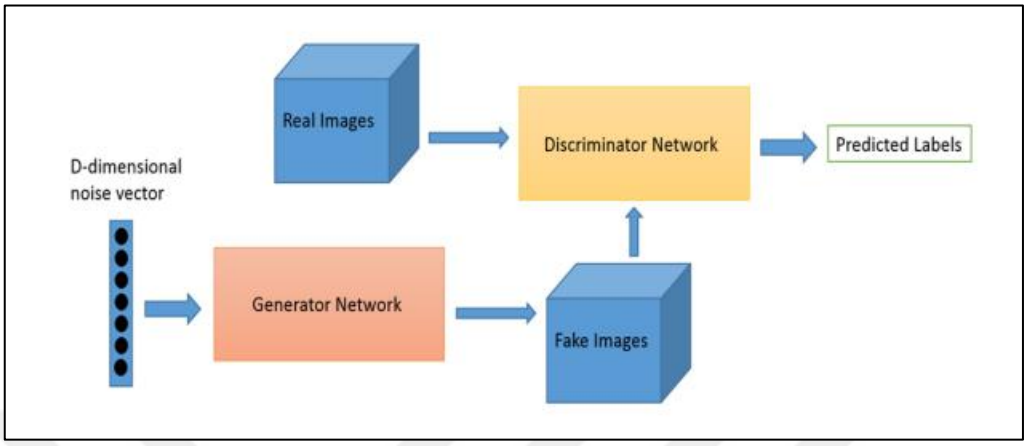

Figure 2.16: The General Architecture of GAN. [70]

2.4.7 Transformer Model

In comparison to earlier systems, the Transformer model's introduction in 2017 [71] drastically cut training time while still producing cutting-edge outcomes in language translation. It has an encoder-decoder design, where the encoder creates an internal representation from the token vector representations (input embeddings) of the input text (as depicted in Fig 2.17). When mapping to the output sequences, the decoder subsequently makes use of the internal representation (the target language, for instance). The Transformer model featured an architecture element termed attention rather than recurrent layers or convolution layers, in contrast to conventional models at the time. The attention module allows a model to take into account the relationships between every pair of tokens in a sequence and automatically identify the associations that are pertinent to the job at hand. There are many different types of attention modules, but the majority of them—including the one employed by [72] automatically figure out how pairs of tokens interact with one another. For the current prediction job, each input token is given a weighted relevance value based on these interaction patterns, which also enable the model to discover relationships between tokens that are spaced widely apart in the input sequence. Multi head attention is the process of using more than one such attention module simultaneously to enable the model to learn various elements of the connections between input tokens.

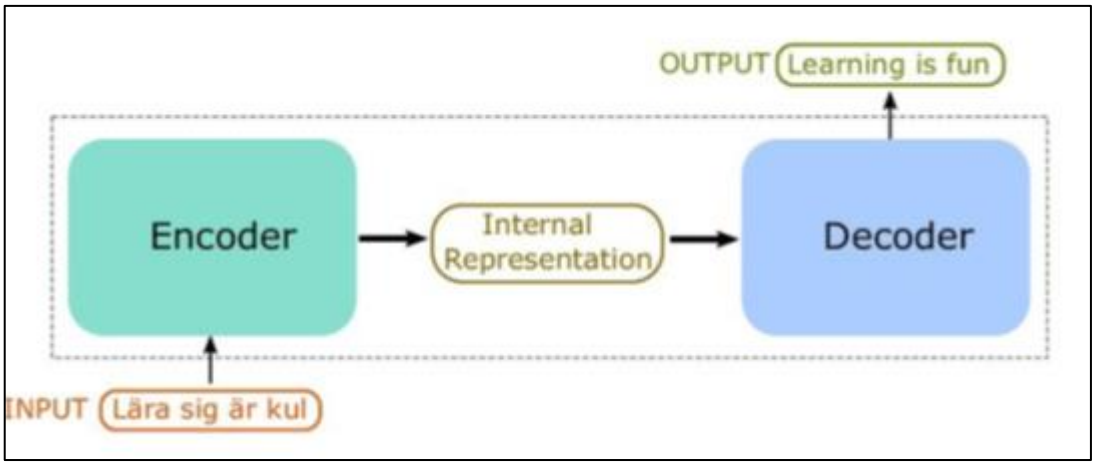

Figure 2.17: Transformer Model. [73]

2.5 CONCLUSION

In conclusion, the objective of creating DL models for malware classification has been well researched in the literature. Various DL classifiers have been used, and performance metrics have been utilized to assess their performance. In the next chapter, the methodology for our research will be introduced.

3. METHODOLOGY

3.1 INTRODUCTION

Artificial intelligence (AI) encloses both ML and DL. ML, in essence, is an AI that can automatically adapt with little assistance from humans. DL is a branch of ML that makes use of artificial neural networks to simulate how the human brain learns. This chapter serves as an introduction to our motivation. We then provide a more thorough analysis of our suggested solution and the steps we took to put our strategy into practice from conception to realization.

3.2 MOTIVATION AND CONTRIBUTION

The motivation for this research is to address the problem of dynamic and evolving malware. Malware is a constantly evolving threat, and new strains of malware are constantly being developed. This means that traditional malware detection methods, such as signature-based detection, are becoming less effective. By using ML algorithms, this research aims to develop a more dynamic and adaptable approach to detecting malware that can adapt to new strains of malware as they are developed.

In addition, this research aims to identify the most effective algorithm for detecting malware in a dataset of binary vectors representing the permissions used by each application, which is important for detecting malware in mobile applications, which are becoming increasingly popular with the widespread use of smartphones. Mobile applications can request a large number of permissions, and it is important to be able to quickly and accurately detect malware among them.

Furthermore, this research also aims to contribute to the field of cybersecurity by providing a new approach to malware detection that can be applied in real-world scenarios. The results of this research can be used to develop new malware detection systems that are more effective and efficient at detecting new strains of malware.

Overall, the motivation of this research is to address the problem of dynamic and evolving malware by developing a more dynamic and adaptable approach to detecting malware using ML/ DL algorithms, which in turn could be applied in real-world scenarios and contribute to the field of cybersecurity.

3.3 RESEARCH APPROACH

In this research, the problem of malware classification was addressed by implementing and comparing the performance of four different ML algorithms. The first algorithm used was KNN, a non-parametric method that is often used for classification and regression problems. KNN is a simple algorithm that assigns a label to an unknown sample based on the majority vote of its k-nearest neighbors.

The second algorithm implemented was the CNN based on a set of blocks that may be used to apply filters to an image to extract the features to classify malware from that image. as for the third algorithm implemented was CNN-LSTM. This algorithm utilizes the strengths of both CNNs and LSTMs to classify malware. CNNs are particularly useful in image and audio processing, while LSTMs are effective in handling sequential data, making CNN-LSTM an attractive choice for malware classification.

The fourth algorithm used was CNN-GRU. Like the CNN-LSTM, this algorithm combines the strengths of CNNs and GRUs for malware classification. CNNs are used for feature extraction and GRUs for sequential data handling.

The implementation of four algorithms was done to investigate their effectiveness in detecting and classifying malware. The performance of these algorithms was evaluated using a dataset of known malware samples and the results were compared to determine the most suitable algorithm for malware classification. This research aims to provide insights into the best ML algorithm for malware classification and contribute to the development of more robust and effective malware detection systems. The proposed model of this research is illustrated in Fig 3.1

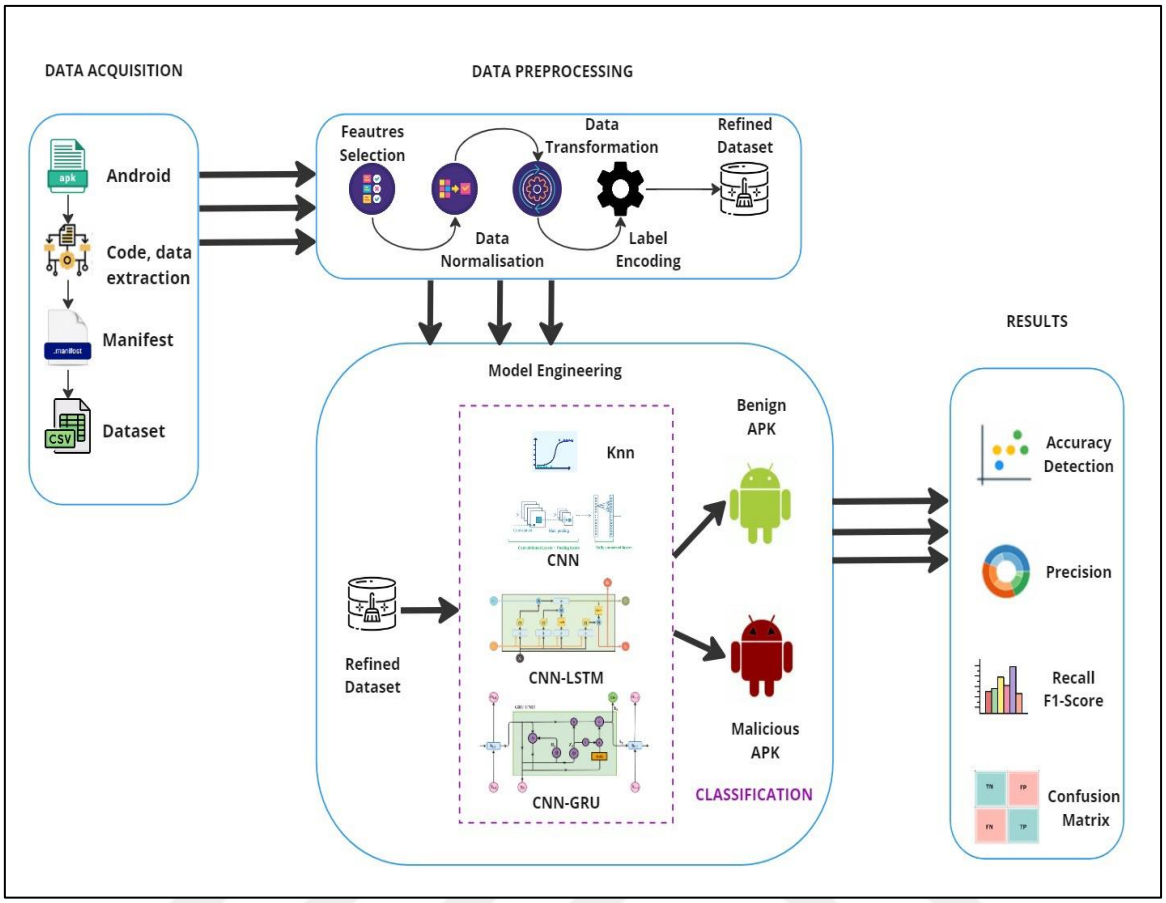

Figure 3.1: Model Approach.

3.3.1 Data Acquisition

A manifest file with the name AndroidManifest.xml is included in an Android APK. Metadata that allows the installation and execution of Android packages is contained in the manifest file. Java code is organized in the suggested manner to extract the manifest file from an Android APK. A Python script has been built to extract important information from the manifest file and generate feature vectors. The created code thoroughly analyzes the manifest file and extracts many aspects (such as permissions, API calls, and filtered intents). We found an online dataset with already data acquisition but the general steps to generate from the APK file are present in Fig 3.1.

3.3.2 Dataset

The proposed methodology for this research project involves using the Dataset malware/ benign permissions Android as the basis for evaluating the performance of the malware Classification approach. The dataset will be used to analyze the permissions used by Android applications and classify them as either malware or benign. The data is represented as a binary vector, with 1 indicating that permission is used and 0 indicating that it is not used. The samples in the dataset are also labeled as either malware (Type 1) or non-malware (Type 0). In ML, it is common to use datasets like this to train a model to classify items based on certain characteristics or features.

The model will be trained and tested on the dataset and the performance will be evaluated using various metrics such as ACC, PREC, recall, and F1 score. The goal is to develop an effective and efficient malware classification approach that can accurately identify and classify malware on Android devices. The dataset is accessible in Kaggle:
https://www.kaggle.com/datasets/xwolf12/datasetandroidpermissions

3.3.3 Data Pre-Processing

Data preprocessing is an important phase of ML as it helps to prepare the dataset for training by cleaning and transforming the data into a format that can be easily understood and processed by the model. The four main steps of data preprocessing are featuring selection, data normalization, data transformation, and label encoding.

a. Feature selection: reduces the complexity of the dataset and improves the performance of the model by selecting a subset of pertinent characteristics from it.
b. Data normalization: is the process of scaling the data so that all features have the same range. This helps to avoid issues such as one feature having a much larger scale than others, which can skew the results.
c. Data transformation**:** is the process of converting the data into a format that is suitable for the model. This can include things such as encoding categorical variables or converting
   data into a binary format.

d. Label encoding**:** is the process of converting the labels into a numerical format, typically 0 and 1. This is done so that the model can understand the labels and make predictions based on them.

In the case of the proposed dataset, malware/benign permissions Android, it has been stated that the dataset is already preprocessed and does not require these four steps. The data is represented as a binary vector, with 1 indicating that a permission is used and 0 indicating that it is not used. The samples in the dataset are also labeled as either malware (Type 1) or non-malware (Type 0), meaning that the label encoding step has already been done. It is important to check if the dataset needs these preprocessing steps or not, before using the dataset for training. In this case, it seems that the dataset is ready to be used for training the malware classification model.

3.4 METHODOLOGY OF RESEARCH

As Fig 3.1 shows, we use four classification methods, KNN is the ML approach, while CNN, CNN-LSTM, and CNN-GRU are the DL approaches.

3.4.1 ML

a. KNN

k-nearest neighbors (KNN) model to make predictions on a test dataset. The KNN model has already been trained on a dataset and the" predict" method is being called on this model with the test dataset as an argument, the predictions made by the KNN model will be binary values, where 1 indicates that a given application is malware and 0 indicates that it is benign. The output of this code is a variable" KNN pred" that contains the predictions made by the KNN model as the Fig 3.2 shows.

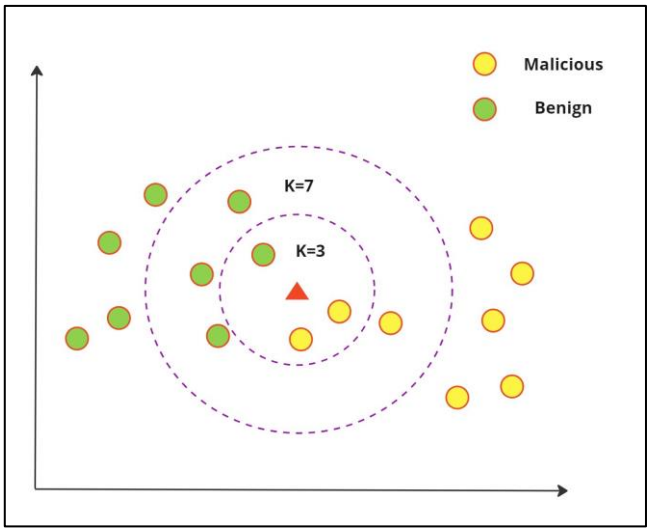

Figure 3.2: KNN Approach.

3.4.2 DL

b. CNN

CNN model using the Keras library. The model is being created using the Sequential API and it is composed of multiple layers. The first layer is a 1D convolutional layer with 128 filters, a kernel size of 1, and a ReLU activation FUN. The input shape of this layer is the shape of the training data, with an additional dimension of 1 added. Then, the model has added several 1D convolutional layers, with 64 and 32 filters respectively, each with (1,) kernel size and ReLU activation FUN.

Each convolutional layer is followed by a 1D max pooling layer with pool size of (1,). Then there is a dropout layer with a rate of 0.25 that is used to prevent overfitting. After flatten the model, there is a final dense layer with 1 unit and sigmoid activation FUN. The model is then compiled using the Adam optimizer, binary cross-entropy loss FUN, and ACC as a metric.

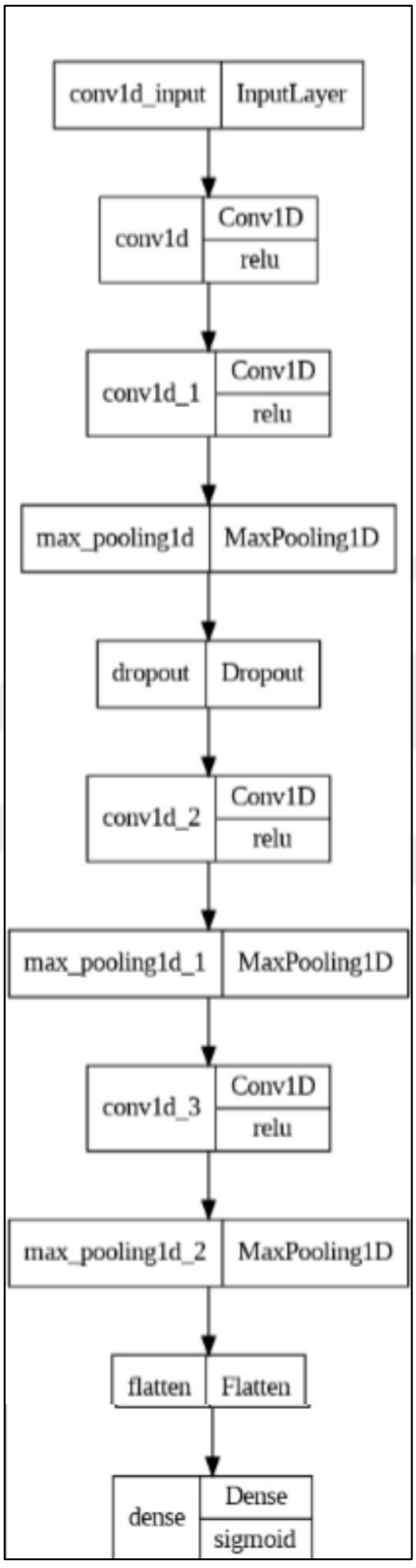

Figure 3.3: CNN Approach**.**

Activity FUN

ReLU

The formula for the Rectified Linear Unit (ReLU) activation FUN is as follows 3.1:

$$f(x) = \max(0,1) \tag{3.1}$$

Sigmoid

The softmax FUN is a commonly used activation FUN in neural networks, particularly in the output layer of a multi-class classification model. The FUN maps the output of the final layer of neurons to a probability distribution over the possible classes.

The formula for the softmax FUN is as follows 3.2:

$$\sigma(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \tag{3.2}$$

where *z* is the input vector, *i* is the index of the current output neuron, *K* is the total number of output neurons, and *e* is the base of the natural logarithm.

Optimization FUN

A popular stochastic gradient descent optimization technique in DL and ML is called Adam (Adaptive Moment Estimation). It is an extension of the traditional stochastic gradient descent method (SGD) that incorporates the benefits of AdaGrad and RMSprop, two further extensions of SGD.

Adam adjusts the learning rate for each parameter using a mix of gradient information, moving averages of the gradient, and squared gradient. Additionally, it has a bias correction term to take into account the initial high learning rates, which is very helpful when deep neural networks are first being trained.

c. CNN-LSTM

Sequential neural network model using the Keras library. It starts by adding a 1D convolutional layer with 32 filters, an input shape matching the shape of the training data, a kernel size of 8, a ReLU activation FUN, a stride of 1, and padding set to" same".

Then it adds a sequence of four LSTM layers with 10 units each, with the option of returning sequences enabled. Between each LSTM layer, there is a dropout layer with a dropout rate

of 0.2, which is used to prevent overfitting by randomly dropping some neurons during the training.

Finally, it adds a dense layer with a single output unit and a sigmoid activation FUN. This is the output layer of the model, which will produce a probability of the input being malware or benign.

The model is then compiled with a binary cross-entropy loss FUN, Adam optimizer and metrics of ACC as Fig 3.4 shows.

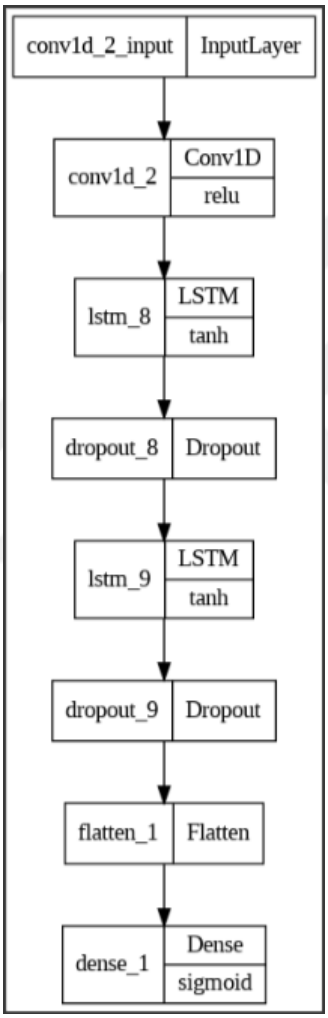

Figure 3.4: CNN-LSTM Approach.

CNN-GRU

A sequential model using the Keras library. It starts with a convolutional 1D layer with 32 filters, a kernel size of 8, a ReLU activation FUN, a stride of 1, and padding set to" same". This is followed by a Gated Recurrent Unit (GRU) layer with 10 units and return-sequences set to True. This is followed by a dropout layer with a dropout rate of 0.2. This pattern of a

GRU layer followed by a dropout layer is repeated three times. The last GRU layer does not have return-sequences set to True. The model ends with a flatten layer followed by a dense layer with 1 unit and sigmoid activation FUN. The model is then compiled using binary cross entropy as the loss FUN, Adam as the optimizer and ACC as the metric.

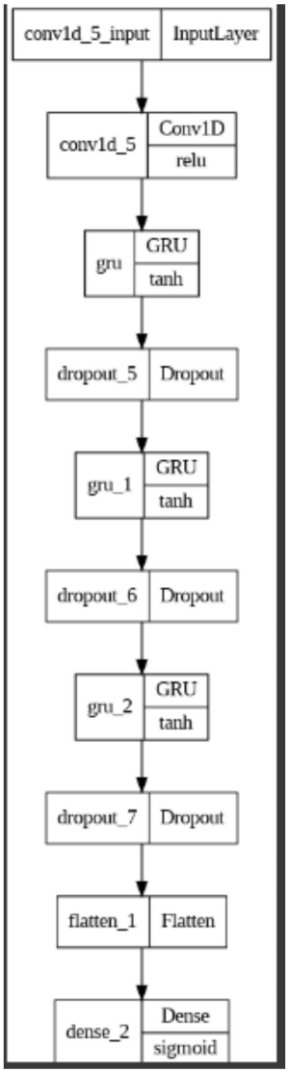

Figure 3.5: CNN-GRU Approach.

3.5 EVALUATION METRICS

An ML model's performance is evaluated using evaluation metrics.

Following are some typical assessment metrics:

a. ACC

The proportion of correct predictions made by the model. The formula for ACC is:

ACC = Number of correct predictions/ Total number of predictions (3.3)

b. PREC

The proportion of true positive predictions out of all positive predictions made by the model. The formula for PREC is:

$$\text{PREC} = \text{True Positives} \setminus \text{True Positives} + \text{False Positives} \quad (3.4)$$

c. REC (also known as Sensitivity or TPR)

The proportion of true positive predictions out of all actual positive observations. The formula for REC is:

$$\text{REC} = \text{True Positives} \setminus \text{True Positives} + \text{False Negatives} \quad (3.5)$$

d. F1 Score

ACC and memory through harmony. It provides balance between REC and ACC. The F1 Score formula is:

$$\text{F1 Score} = 2 * (\text{PREC} * \text{REC} \setminus \text{PREC} + \text{Recall}) \quad (3.6)$$

e. Confusion matrix

The number of true positives, true negatives, false positives, and false negatives are displayed in a table. It gives a clear idea about how many observations are classified correctly and incorrectly.

3.6 CONCLUSION

Presented in this chapter is our solution. We began by outlining our contribution and showing how it differs from the works discussed in the previous chapter. We presented the architecture of our solution and then we implemented the various phases of the development pipeline.

4. PERFORMANCE EVALUATION

4.1 INTRODUCTION

In this chapter, we will present the experimental findings of malware classification using ML and DL algorithms.

The results from the various implemented classifiers will then be provided in terms of ACC, PREC, REC and F1 score

4.2 RESULTS

Using assessment metrics like ACC, PREC, recall, and F1-score, we will go over the results of each ML technique employed in this section. These metrics provide crucial details about the benefits and drawbacks of each technique and are typically used to assess a classifier's or predictor's performance.

4.3 MALWARE CLASSIFICATION USING ML

4.3.1 KNN

The provided array seems to represent the results of a binary classification problem (Fig 4.1). The two columns represent the predicted probabilities of the positive class (e.g. malware) and the negative class (e.g. benign).

In the first row, the predicted probability of the positive class (malware) is 0.945, while the probability of the negative class (benign) is 0.0545. This indicates that the model is confident that the sample belongs to the positive class (malware).

In the second row, the predicted probability of the positive class is 0.2 and the probability of the negative class is 0.8, meaning that the model is less confident and believes that the sample belongs to the negative class (benign).

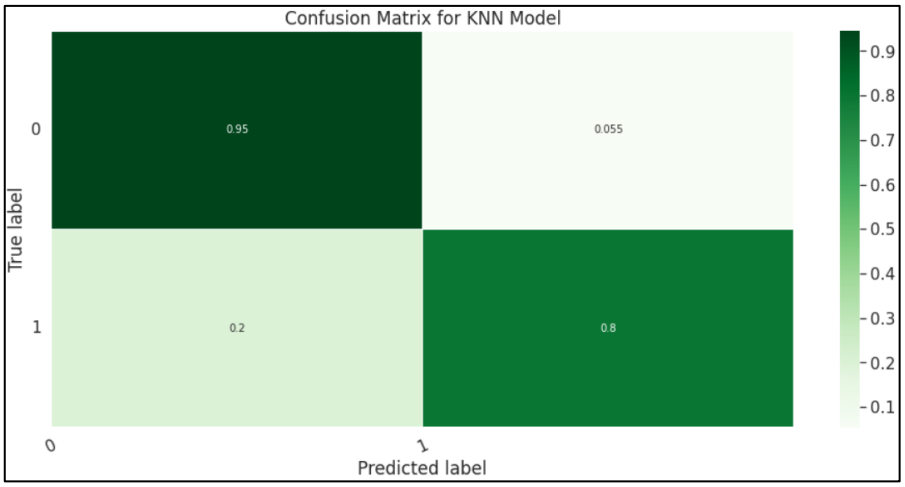

Figure 4.1: KNN Confusion Matrix.

The classification (Fig 4.2) report summarizes the performance of a classification model. The PREC, REC, and F1-S values are metrics that help us to evaluate the ACC of the model.

In this case, the PREC of class 0 is 0.80, meaning 80% of the time the model correctly predicted the positive class, i.e., class 0. The REC of class 0 is 0.95, meaning the model was able to correctly identify 95% of the positive class. The F1-S of class 0 is 0.87, which is the harmonic mean of PREC and recall.

Similarly, for class 1, the PREC is 0.95, the REC is 0.80, and the F1-S is 0.87. The ACC of the model is 0.87, meaning that 87% of the predictions made by the model were correct.

	precision	recall	f1-score	support
0	0.80	0.95	0.87	55
1	0.95	0.80	0.87	65
accuracy			0.87	120
macro avg	0.87	0.87	0.87	120
weighted avg	0.88	0.87	0.87	120

Figure 4.2: KNN Metrics.

4.4 MALWARE CLASSIFICATION USING DL

4.4.1 CNN

This result (Fig 4.3) indicates that after training the CNN for 50 epochs, the model achieved a low loss value of 0.0532 and high ACC value of 0.9856. The low loss value suggests that the model is able to correctly predict the desired outputs for a majority of the input samples in the training set. The high ACC value further confirms that the model is making correct predictions for a large proportion of the input data. This result is a good indication that the model is capable of generalizing to new data.

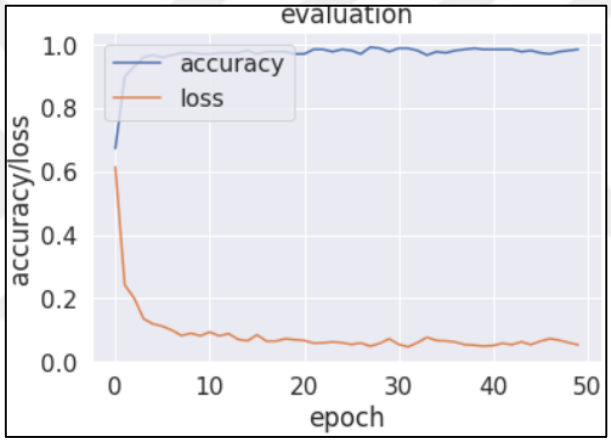

Figure 4.3: CNN-Loss vs Accuracy.

This is a confusion matrix (Fig 4.4), which shows the performance of a binary classification model. It displays the number of correct and incorrect predictions made by the model.

In this matrix, the rows correspond to the true class labels, while the columns correspond to the predicted class labels. For example, the first row shows the results for the class 0. Out of the total predictions for class 0, 96.36% of them were correct, and 3.64% of them were incorrect. Similarly, for class 1, 88% of the predictions were correct, and 12% of them were incorrect.

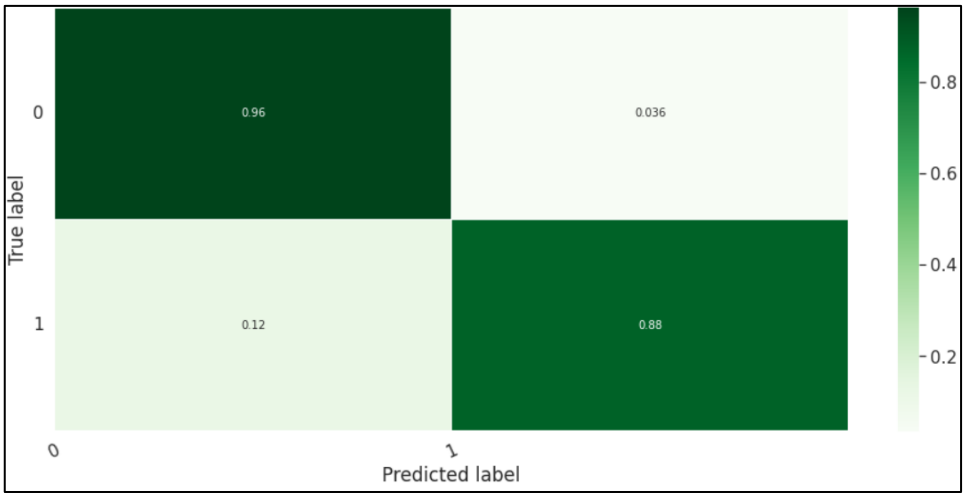

Figure 4.4: CNN Confusion Matrix.

In this report (Fig 27), the model achieved an ACC of 0.92 which means that the model correctly classified 92% of the samples in the test set. The PREC for class 0 is 0.87, which means that out of all the positive predictions made for class 0, 87% were true positive. The REC for class 0 is 0.96, which means that the model correctly identified 96% of the actual positive samples of class 0. The PREC for class 1 is 0.97 and REC is 0.88. The weighted average F1-Sis 0.92, which takes into account the imbalance in the number of samples between the two classes.

Classification report :

	precision	recall	f1-score	support
0	0.87	0.96	0.91	55
1	0.97	0.88	0.92	65
accuracy			0.92	120
macro avg	0.92	0.92	0.92	120
weighted avg	0.92	0.92	0.92	120

Figure 4.5: CNN Metrics.

4.4.2 CNN-LSTM

This result (Fig 4.6), indicates the performance of a CNN-LSTM model on a certain dataset. The model achieved an ACC of 96.76% which means that out of the total number of samples in the dataset, 96.76% of them were correctly classified by the model. The loss is 0.1056 which is the value of the loss FUN used during training of the model. A low value of loss indicates that the model's prediction is close to the ground truth. So, the result of a low loss and high ACC suggest that the model has performed well.

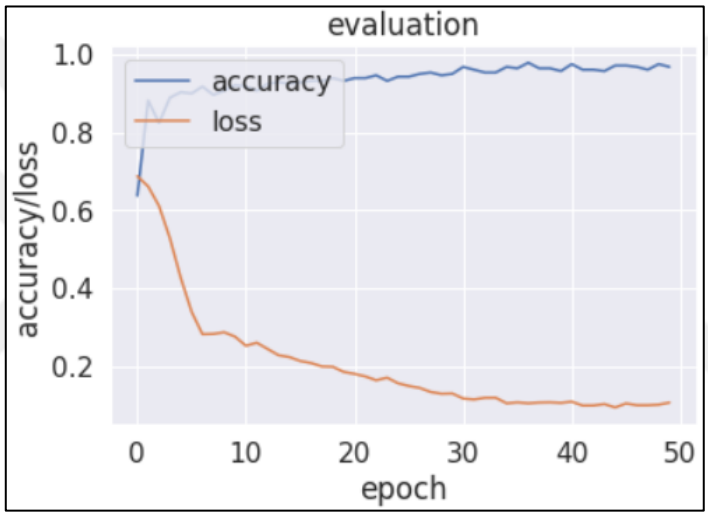

Figure 4.6: CNN-LSTM_Loss vs ACC.

The entries in the matrix (Fig 4.7), are the number of instances that are correctly and incorrectly classified by the model. For example, the first entry in the first row shows that 0.96 of the instances that were actually class 0 were classified as class 0 by the model. The second entry in the first row shows that 0.036 of the instances that were actually class 0 were classified as class 1 by the model (i.e., false positive).

Similarly, the first entry in the second row shows that 0.15 of the instances that were actually class 1 were classified as class 0 by the model (i.e., false negative). The second entry in the second row shows that 0.85 of the instances that were actually class 1 were classified as class 1 by the model.

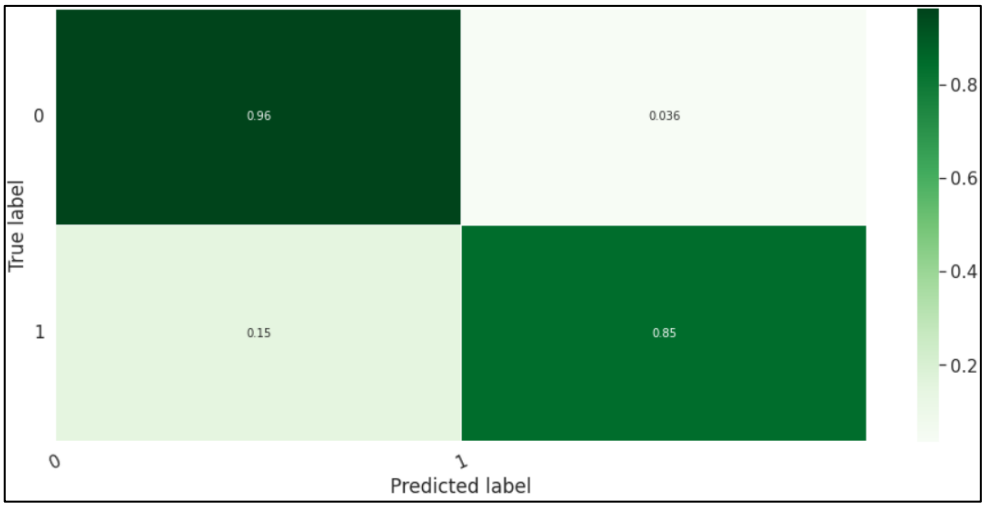

Figure 4.7: CNN-LSTM Metrics.

This is a classification (Fig 4.8), report for a binary classification problem. The report provides several evaluation metrics for the binary classifier including PREC, recall, f1-score, and ACC. The PREC of class 1 (malware) is 0.96, which means that 96% of the positive predictions made by the classifier are true malware samples. The PREC of class 0 (non-malware) is 0.84. The REC of class 1 (malware) is 0.85, which means that 85% of the malware samples in the ground truth are correctly identified as malware. The REC of class 0 (non-malware) is 0.96. The F1-Sof class 1 (malware) is 0.90 and the F1-Sof class 0 (non-malware) is 0.90.

Finally, the ACC of the classifier is 0.90, which means that 90% of the samples are correctly classified. However, ACC should be used with caution, especially in imbalanced datasets, as it can be misleading. In such cases, it is better to use the PREC, recall, or F1-Sto evaluate the performance of the classifier.

Classification report :

	precision	recall	f1-score	support
0	0.84	0.96	0.90	55
1	0.96	0.85	0.90	65
accuracy			0.90	120
macro avg	0.90	0.90	0.90	120
weighted avg	0.91	0.90	0.90	120

Figure 4.8: CNN-LSTM Metrics.

4.4.3 CNN-GRU

This result (Fig 4.9), shows that the CNN-GRU model was trained for 50 epochs and achieved a loss value of 0.1056 and ACC of 0.9676. A loss value close to 0 indicates that the model is well-fit to the training data and a high ACC score suggests that the model is making correct predictions for a large portion of the testing data. This result is a good indication that the model is performing well and is capable of making accurate predictions on unseen data.

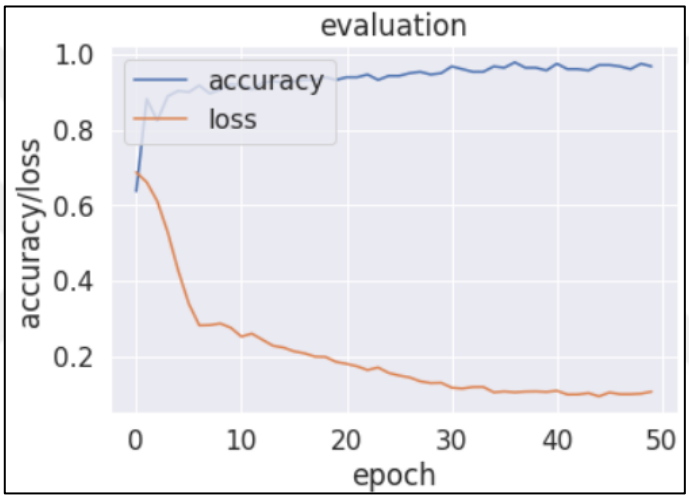

Figure 4.9: CNN-GRU_Loss vs ACC.

The values Confusion matrix (Fig 4.10), in the diagonal elements (0.945 and 0.923) represent the correct classifications and the other elements represent the misclassifications.

The high values in the diagonal indicate that the classifier is making mostly correct predictions, which implies that the ACC is high. However, this result should be analyzed in conjunction with other evaluation metrics to obtain a comprehensive understanding of the classifier's performance.

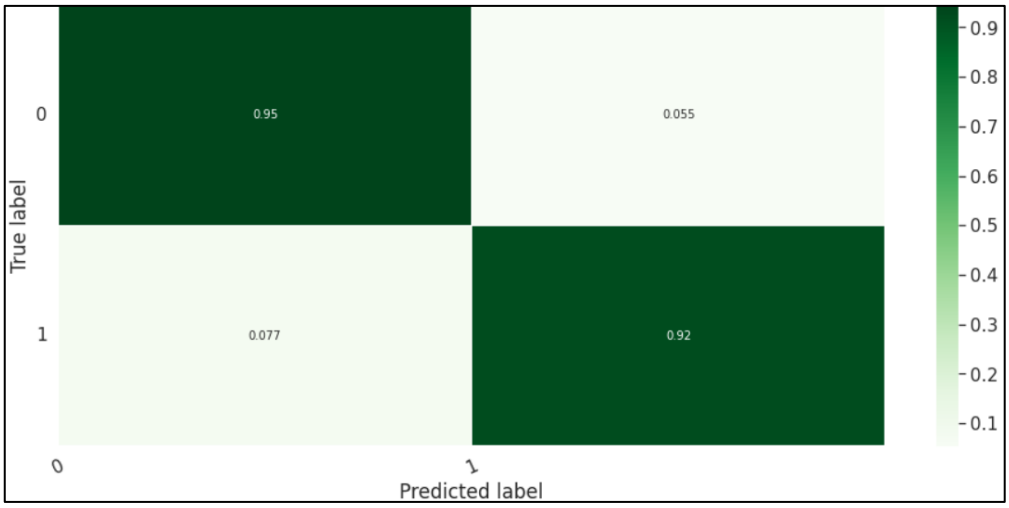

Figure 4.10: CNN-GRU Confusion Matrix.

The result (Fig 4.11), of the model shows an ACC of 93%. The model is performing well for both classes with a PREC of around 91% for class 0 and 95% for class 1. The REC for class 0 is 95% and for class 1 is 92%. The overall weighted average F1-Sis 93%.

	precision	recall	f1-score	support
0	0.91	0.95	0.93	55
1	0.95	0.92	0.94	65
accuracy			0.93	120
macro avg	0.93	0.93	0.93	120
weighted avg	0.93	0.93	0.93	120

Figure 4.11: CNN-GRU Metrics.

4.5 COMPARSION RESULTS

In malware classification using KNN, the provided array represents predicted probabilities of positive (malware) and negative (benign) class. The PREC, REC, and F1-S are used to evaluate the ACC of the model. In the classification report, the ACC is 0.87 and PREC, REC, and F1-S values are also given for both classes.

In malware classification using DL and CNN, the model achieved an ACC of 0.9856 with low loss value after training for 50 epochs. The confusion matrix shows the number of

correct and incorrect predictions made by the model. In the classification report, PREC, recall, and F1-Svalues are given for both classes along with an overall ACC of 0.91.

In malware classification using CNN-LSTM, the model achieved an ACC of 90% with low loss value. The confusion matrix displays the number of instances correctly and incorrectly classified. The classification report includes PREC, REC, f1-S, and ACC for both classes.

we can conclude also that the CNN-GRU model has performed well on the given dataset, with a high ACC score of 0.9676 and a low loss value of 0.1056. The values in the confusion matrix and the other evaluation metrics (PREC, REC, and f1-S) also suggest that the model is making mostly correct predictions and has a good balance between PREC and REC for both classes as shown at the table below 4.1. However, it is important to consider other evaluation metrics and perform further analysis to have a comprehensive understanding of the model's performance.

Table 4.1: The Results Of The Comparison Between The Four Algorithms.

Methods	ACC	PREC	Recall	F1-score	Loss
KNN	87%	87%	87%	87%	-
CNN	91%	92%	92%	92%	0.0532
CNN-LSTM	90%	90%	90%	90%	0.1056
CNN-GRU	95%	92%	93%	94%	0.1056

4.6 CONCLUSION

In this chapter, we discussed the experimental findings from the application of DL and ML models for detecting web application attacks. We used KNN models for ML techniques. Moreover, we used a featureless CNN, CNN-LST and CNN-GRU for the DL method.

5. CONCLUSION

In this work, we used a ML classifier approach to categorize malware. KNN and three DL methods were utilized as classifiers. LSTM CNN, CNN-GRU, and CNN

The results showed that the ACC, PREC, REC, and F1-S of each classifier were good. However, those approaching received the best ratings across all four parameters, suggesting that they could be the most successful in classifying malware.

Overall, the study's findings indicate that ML approaches might be useful for classifying malware.

REFERENCES

[1] Yanfang Ye et al. “A survey on malware detection using data mining techniques”. In: ACM Computing Surveys (CSUR) 50.3 (2017), pp. 1–40.

[2] Daniele Ucci, Leonardo Aniello, and Roberto Baldoni. “Survey of ML techniques for malware analysis”. In: Computers & Security 81 (2019), pp. 123–147.

[3] Ferhat Ozgur Catak and Ahmet Faruk Yazı. “A benchmark API call dataset for windows PE malware classification”. In: arXiv preprint arXiv:1905.01999 (2019).

[4] Ekta Gandotra, Divya Bansal, and Sanjeev Sofat. “Malware analysis and classification: A survey”. In: Journal of Information Security 2014 (2014).

[5] Abdurrahman Pekta, s and Tankut Acarman. “Classification of malware families based on runtime behaviors”. In: Journal of information security and applications 37 (2017), pp. 91–100.

[6] Cho Cho San, Mie Mie Su Thwin, and Naing Linn Htun. “Malicious software family classification using ML multi-class classifiers”. In: Computational Science and Technology: 5th ICCST 2018, Kota Kinabalu, Malaysia, 29-30 August 2018. Springer. 2019, pp. 423–433.

[7] Andrii Shalaginov et al. “ML aided static malware analysis: A survey and tutorial”. In: Cyber threat intelligence (2018), pp. 7–45.

[8] Baoguo Yuan et al. “Byte-level malware classification based on markov images and DL”. In: Computers & Security 92 (2020), p. 101740.

[9] Kyoung Soo Han et al. “Malware analysis using visualized images and entropy graphs”. In: International Journal of Information Security 14 (2015), pp. 1–14.

[10] Liu Liu and Baosheng Wang. “Malware classification using gray-scale images and ensemble learning”. In: 2016 3rd international conference on systems and informatics (ICSAI). IEEE. 2016, pp. 1018–1022.

[11] Sang Ni, Quan Qian, and Rui Zhang. “Malware identification using visualization images and DL”. In: Computers & Security 77 (2018), pp. 871–885.

[12] Songqing Yue. “Imbalanced malware images classification: a CNN based approach”. In: arXiv preprint arXiv:1708.08042 (2017).

[13] Andrew H Sung et al. “Static analyzer of vicious executables (save)”. In: 20th Annual Computer Security Applications Conference. IEEE. 2004, pp. 326–334.

[14] Mihai Christodorescu and Somesh Jha. Static analysis of executables to detect malicious patterns. Tech. rep. Wisconsin Univ-Madison Dept of Computer Sciences, 2006.

[15] S Momina Tabish, M Zubair Shafiq, and Muddassar Farooq. “Malware detection using statistical analysis of byte-level file content”. In: Proceedings of the ACM SIGKDD Workshop on Cybersecurity and Intelligence Informatics. 2009, pp. 23–31.

[16] Igor Santos et al. “Opcode sequences as representation of executables for datamining-based unknown malware detection”. In: information Sciences 231 (2013), pp. 64–82.

[17] Babak Yadegari et al. “A generic approach to automatic deobfuscation of executable code”. In: 2015 IEEE Symposium on Security and Privacy. IEEE. 2015, pp. 674–691.

[18] Jake Drew, Tyler Moore, and Michael Hahsler. “Polymorphic malware detection using sequence classification methods”. In: 2016 IEEE Security and Privacy Workshops (SPW). IEEE. 2016, pp. 81–87.

[19] Igor Popov. “Malware detection using ML based on word2vec embeddings of machine code instructions”. In: 2017 Siberian symposium on data science and engineering (SSDSE). IEEE. 2017, pp. 1–4.

[20] Lakshmanan Nataraj et al. “Malware images: visualization and automatic classification”. In: Proceedings of the 8th international symposium on visualization for cyber security. 2011, pp. 1–7.

[21] Olga Russakovsky et al. “Imagenet large scale visual recognition challenge”. In: International journal of computer vision 115 (2015), pp. 211–252.

[22] Seyyed Hossein Hasanpour et al. “Towards principled design of deep convolutional networks: Introducing simpnet”. In: arXiv preprint arXiv:1802.06205 (2018).

[23] Hyrum S Anderson and Phil Roth. “Ember: an open dataset for training static pe malware ML models”. In: arXiv preprint arXiv:1804.04637 (2018).

[24] Mansour Ahmadi et al. “Novel feature extraction, selection and fusion for effective malware family classification”. In: Proceedings of the sixth ACM conference on data and application security and privacy. 2016, pp. 183–194.

[25] Chan Woo Kim. “Ntmaldetect: A ML approach to malware detection using native api system calls”. In: arXiv preprint arXiv:1802.05412 (2018).

[26] Taiwo Oladipupo Ayodele. “ML overview”. In: New Advances in Machine Learning 2 (2010), pp. 9–18.

[27] Leo Breiman. “Random forests”. In: ML 45 (2001), pp. 5–32.

[28] Mourad Azhari et al. “Adaptation of the random forest method: solving the problem of pulsar search”. In: Proceedings of the 4th International Conference on Smart City Applications. 2019, pp. 1–6.

[29] Mei-Hung Chiu et al. “The use of facial micro-expression state and Tree-Forest Model for predicting conceptual-conflict based conceptual change”. In: Chapter Title & Authors (2016), p. 2016.

[30] Marvin Minsky. “Steps toward artificial intelligence”. In: Proceedings of the IRE 49.1 (1961), pp. 8–30.

[31] J. inez Otzeta et al. “Naive + Naive = Smart Bayes?” In: (Jan. 2023).

[32] Loubna Benabbou. “Cont˜ ibutions `a la classification supervis´ee multi-classes et rnulticrit`ere en aide `a la d´ecision”. In: (2009).

[33] Bernhard E Boser, Isabelle M Guyon, and Vladimir N Vapnik. “A training algorithm for optimal margin classifiers”. In: Proceedings of the fifth annual workshop on Computational learning theory. 1992, pp. 144–152.

[34] Corinna Cortes and Vladimir Vapnik. “Support vector machine”. In: ML 20.3 (1995), pp. 273–297.

[35] Khouma Ousmane et al. “Novel Classification Method of Spikes Morphology in EEG Signal Using ML”. In: Procedia Computer Science 148 (Jan. 2019), pp. 70–79. doi: 10.1016/j.procs.2019.01.010.

[36] Muhammet Fatih Ak. “A Comparative Analysis of Breast Cancer Detection and Diagnosis Using Data Visualization and ML Applications”. In: Healthcare 8 (Apr. 2020), p. 111. doi: 10.3390/healthcare8020111.

[37] Tianqi Chen and Carlos Guestrin. “Xgboost: A scalable tree boosting system”. In: Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining. 2016, pp. 785–794.

[38] Leo Breiman. Arcing the edge. Tech. rep. Citeseer, 1997.

[39] Jerome H Friedman. “Stochastic gradient boosting”. In: Computational statistics & data analysis 38.4 (2002), pp. 367–378.

[40] Tao Zhang et al. “Improving convection trigger FUN s in deep convective parameterization schemes using ML”. In: Journal of Advances in Modeling Earth Systems 13.5 (2021), e2020MS002365.4

[41] Sun-ChongWang. Interdisciplinary computing in Java programming. Vol. 743. Springer Science & Business Media, 2003.

[42] Md Shopon, Nabeel Mohammed, and Md Anowarul Abedin. “Image augmentation by blocky artifact in deep convolutional neural network for handwritten digit recognition”. In: 2017 IEEE International Conference on Imaging, Vision & Pattern Recognition (icIVPR). IEEE. 2017, pp. 1–6.

[43] Antoine Bordes et al. “Joint learning of words and meaning representations for open-text semantic parsing”. In: Artificial intelligence and statistics. PMLR. 2012, pp. 127–135.

[44] Dan C Cire, san, Ueli Meier, and J¨urgen Schmidhuber. “Transfer learning for Latin and Chinese characters with deep neural networks”. In: The 2012 international joint conference on neural networks (IJCNN). IEEE. 2012, pp. 1–6.

[45] Jimmy Ren and Li Xu. “On vectorization of deep convolutional neural networks for vision tasks”. In: Proceedings of the AAAI Conference on Artificial Intelligence. Vol. 29. 1. 2015.

[46] Tomas Mikolov et al. “Distributed representations of words and phrases and them compositionality”. In: Advances in neural information processing systems 26 (2013).

[47] Dan Ciregan, Ueli Meier, and J¨urgen Schmidhuber. “Multi-column deep neural networks for image classification”. In: 2012 IEEE conference on computer vision and pattern recognition. IEEE. 2012, pp. 3642–3649.

[48] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “Imagenet classification with deep convolutional neural networks”. In: Communications of the ACM 60.6 (2017), pp. 84–90.

[49] Li Deng. “A tutorial survey of architectures, algorithms, and applications for deep learning”. In: APSIPA transactions on Signal and Information Processing 3 (2014), e2.

[50] Yann LeCun et al. “Gradient-based learning applied to document recognition”. In: Proceedings of the IEEE 86.11 (1998), pp. 2278–2324.

[51] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “Imagenet classification with deep convolutional neural networks”. In: Communications of the ACM 60.6 (2017), pp. 84–90.

[52] Yanming Guo et al. “DL for visual understanding: A review”. In: Neurocomputing 187 (2016), pp. 27–48.

[53] Cheng-Yuan Liou et al. “Autoencoder for words”. In: Neurocomputing 139 (2014), pp. 84–96.

[54] Geoffrey E Hinton and Ruslan R Salakhutdinov. “Reducing the dimensionality of data with neural networks”. In: science 313.5786 (2006), pp. 504–507.

[55] Jie Zhang et al. “Coarse-to-fine auto-encoder networks (cfan) for real-time face alignment”. In: Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part II 13. Springer. 2014, pp. 1–16.

[56] Yingbo Zhou et al. “Is joint training better for deep auto-encoders?” In: arXiv preprint arXiv:1405.1380 (2014).

[57] Xin Yao. “Evolving artificial neural networks”. In: Proceedings of the IEEE 87.9 (1999), pp. 1423–1447.

[58] Wim De Mulder, Steven Bethard, and Marie-Francine Moens. “A survey on the application of recurrent neural networks to statistical language modeling”. In: Computer Speech & Language 30.1 (2015), pp. 61–98.

[59] Chung-Ming Kuan and Tung Liu. “Forecasting exchange rates using feedforward and recurrent neural networks”. In: Journal of applied econometrics 10.4 (1995), pp. 347–364.

[60] Yanan Guo et al. “El Ni˜no Index Prediction Using DL with Ensemble Empirical Mode Decomposition”. In : Symmetry 12 (June 2020), p. 893. doi: 10. 3390/sym12060893.

[61] Iram Bibi et al. “A Dynamic DL-Driven Architecture to Combat Sophisticated Android Malware”. In: IEEE Access PP (July 2020), pp. 1–1. doi: 10.1109/ACCESS. 2020.3009819.

[62] Zhi-Hua Zhou. “Ensemble Learning”. In: Encyclopedia of Biometrics. Ed. by Stan Z. Li and Anil Jain. Boston, MA: Springer US, 2009, pp. 270–273. isbn: 978-0-387-73003-5. doi: 10.1007/978-0-387-73003-5_293. Url : https://doi.org/10.1007/978-0-387-73003-5_293.

[63] Cha Zhang and Yunqian Ma. Ensemble ML: methods and applications. Springer, 2012.

[64] Elnaz Sharghi, Vahid Nourani, and Nazanin Behfar. “Earthfill dam seepage analysis using ensemble artificial intelligence-based modeling”. In: Journal of Hydroinformatics 20.5 (2018), pp. 1071–1084.

[65] Mohammad Zounemat-Kermani et al. “Ensemble data mining modeling in corrosion of concrete sewer: A comparative study of network-based (MLPNN & RBFNN) and tree-

based (RF, CHAID, & CART) models". In: Advanced Engineering Informatics 43 (2020), p. 101030.

[66] Mohammad Zounemat-Kermani et al. "Neurocomputing in surface water hydrology and hydraulics: A review of two decades retrospective, current status and future prospects". In: Journal of Hydrology 588 (2020), p. 125085.

[67] Donghwan Kim et al. "Ensemble learning regression for estimating river discharges using satellite altimetry data: Central Congo River as a Test-bed". In: Remote sensing of environment 221 (2019), pp. 741–755.

[68] Mohammad Zounemat-Kermani et al. "Ensemble ML paradigms in hydrology: A review". In: Journal of Hydrology 598 (2021), p. 126266.

[69] Ian Goodfellow et al. "Generative adversarial networks". In: Communications of the ACM 63.11 (2020), pp. 139–144.

[70] Hamed Alqahtani, Manolya Kavakli-Thorne, and Gulshan Kumar. "Applications of generative adversarial networks (gans): An updated review". In: Archives of Computational Methods in Engineering 28 (2021), pp. 525–552.

[71] Ashish Vaswani et al. "Attention is all you need". In: Advances in neural information processing systems 30 (2017).

[72] Jun Cheng et al. "BERTMHC: improved MHC–peptide class II interaction prediction with transformer and multiple instance learning". In: Bioinformatics 37.22 (2021), pp. 4172–4179.

[73] Abel Chandra et al. "Transformer-based DL for predicting protein properties in the life sciences". In: eLife 12 (2023), e82819. 38

[74] Bibi, Iram, et al. "A dynamic DL-driven architecture to combat sophisticated Android malware." IEEE Access 8 (2020): 129600-129612.

[75] Kim, Jinsung, et al. "MAPAS: a practical DL -based android malware detection system." International Journal of Information Security 21.4 (2022): 725-738.

[78] Muzaffar, Ali, et al. "An in-depth review of ML based android malware detection." Computers & Security (2022): 102833.

[79] Liu, Yue, et al. " DL for android malware defences: a systematic literature review." ACM Journal of the ACM (JACM) (2022).

[80] Amer, Eslam, and Shaker El-Sappagh. "Robust DL early alarm prediction model based on the behavioural smell for android malware." Computers & Security 116 (2022): 102670.